Package 'CNEr'

December 23, 2024

Version 1.42.0

Date 2024-04-23

Title CNE Detection and Visualization

Description Large-scale identification and advanced visualization of sets of conserved noncoding elements.

Author Ge Tan <ge_tan@live.com>

Maintainer Ge Tan <ge_tan@live.com>

Imports Biostrings (>= 2.33.4), pwalign, DBI (>= 0.7), RSQLite (>= 0.11.4), GenomeInfoDb (>= 1.1.3), GenomicRanges (>= 1.23.16), rtracklayer (>= 1.25.5), XVector (>= 0.5.4), GenomicAlignments (>= 1.1.9), methods, S4Vectors (>= 0.13.13), IRanges (>= 2.5.27), readr (>= 0.2.2), BiocGenerics, tools, parallel, reshape2 (>= 1.4.1), ggplot2 (>= 2.1.0), poweRlaw (>= 0.60.3), annotate (>= 1.50.0), GO.db (>= 3.3.0), R.utils (>= 2.3.0), KEGGREST (>= 1.14.0)

Depends R (>= 3.4)

Suggests Gviz (>= 1.7.4), BiocStyle, knitr, rmarkdown, testthat, BSgenome.Drerio.UCSC.danRer10, BSgenome.Hsapiens.UCSC.hg38, TxDb.Drerio.UCSC.danRer10.refGene, BSgenome.Hsapiens.UCSC.hg19, BSgenome.Ggallus.UCSC.galGal3

LinkingTo S4Vectors, IRanges, XVector

VignetteBuilder knitr

License GPL-2 | file LICENSE

License_restricts_use yes

URL https://github.com/ge11232002/CNEr

BugReports https://github.com/ge11232002/CNEr/issues

Type Package

biocViews GeneRegulation, Visualization, DataImport

NeedsCompilation yes

LazyData no

Collate GRangePairs-class.R GRangePairs-methods.R Axt-class.R CNE-class.R utils.R ceScan.R plot.R makeGeneDbFromUCSC.R IO-methods.R scoringMatrix.R subAxt-methods.R Axt-methods.R DB.R AssemblyStats.R GRB.R WholeGenomeAlignment.R Ancora.R CNE-methods.R GO.R KEGG.R

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

git_branch RELEASE_3_20

git_last_commit d83a1b6

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-12-23

Contents

`addAncestorGO` *Add ancestor GO IDs*

Description

Given a list of GO IDs, add the corresponding ancestor GO IDs.

Usage

```
addAncestorGO(go)
```

Arguments

`go`	A `list` of GO IDs. The elements of the list can be empty.

Details

The ancestor GO IDs for each GO ID are added to the elements.

Value

A `list` of GO IDs with their ancestor GO IDs.

Note

This function is mainly designed for processing the gff annotation generated from interproscan, where for each gene, a set of GO IDs are assigned. However, for GO enrichment analysis, we need a list of mapping from genes to the GO IDs and their ancestor GO IDs as well.

Author(s)

Ge Tan

Examples

```
## Not run:
  library(GO.db)
  go <- list(c("GO:0005215", "GO:0006810", "GO:0016020"), "GO:0016579")
  addAncestorGO(go)

## End(Not run)
```

axisTrack *Example data for plotting annotation.*

Description

Five annotation tracks for plotting in Gviz.

Usage

```
data(axisTrack)
data(cpgIslands)
data(refGenes)
```

Details

These tracks are based on genome="danRer10", chr = "chr6", start = 24000000, end = 27000000.

Examples

```
data(axisTrack)
data(cpgIslands)
data(refGenes)
```

Axt-class *Class "Axt"*

Description

The Axt S4 object to hold a axt file.

Usage

```
## Constructors:
Axt(targetRanges=GRanges(), targetSeqs=DNAStringSet(),
    queryRanges=GRanges(), querySeqs=DNAStringSet(),
    score=integer(0), symCount=integer(0), names=NULL)

## Accessor-like methods:
## S4 method for signature 'Axt'
targetRanges(x)
## S4 method for signature 'Axt'
targetSeqs(x)
## S4 method for signature 'Axt'
queryRanges(x)
## S4 method for signature 'Axt'
querySeqs(x)
## S4 method for signature 'Axt'
score(x)
## S4 method for signature 'Axt'
symCount(x)
## ... and more (see Methods)
```

Arguments

targetRanges	Object of class "GRanges": The ranges of net alignments on reference genome.
targetSeqs	Object of class "DNAStringSet": The alignment sequences of reference genome.
queryRanges	Object of class "GRanges": The ranges of net alignments on query genome.
querySeqs	Object of class "DNAStringSet": The alignment sequences of query genome.
score	Object of class "integer": The alignment score.
symCount	Object of class "integer": The alignment length.
names	character(): the names of axt alignments.
x	Object of class "Axt": A Axt object.

Details

In 'axt' files and Axt object, the 'targetRanges' also have the alignments on positive strands. However, the 'queryRanges' can have alignments on negative strands, and the coordinates are based on negative strands, which is quite different from the convention in Bioconductor. To convert the coordinates of alignments on the negative strand to the positive strand, use normaliseStrand.

Methods

[signature(x = "Axt", i = "ANY", j = "ANY"): Axt getter

c signature(x = "Axt"): Axt concatenator.

length signature(x = "Axt"): Get the number of alignments.

queryRanges signature(x = "Axt"): Get the ranges of query genome.

querySeqs signature(x = "Axt"): Get the alignment sequences of query genome.

score signature(x = "Axt"): Get the alignment score.

symCount signature(x = "Axt"): Get the alignment lengths.

targetRanges signature(x = "Axt"): Get the ranges of reference genome.

targetSeqs signature(x = "Axt"): Get the alignment sequences of reference genome.

Author(s)

Ge Tan

See Also

readAxt writeAxt subAxt fixCoordinates makeAxtTracks

Examples

```
library(GenomicRanges)
library(Biostrings)
## Constructor
targetRanges <- GRanges(seqnames=c("chr1", "chr1", "chr2", "chr3"),
                        ranges=IRanges(start=c(1, 20, 2, 3),
                                       end=c(10, 25, 10, 10)),
                        strand="+")
targetSeqs <- DNAStringSet(c("ATTTTATGTG", "GGGAAG", "GGGCTTTTG",
                             "TTGTGTAG"))
queryRanges <- GRanges(seqnames=c("chr1", "chr10", "chr10", "chr20"),
```

```
                        ranges=IRanges(start=c(1, 25, 50, 5),
                                       end=c(10, 30, 58, 12)),
                        strand="+")
querySeqs <- DNAStringSet(c("ATTTAAAGTG", "GGAAAA", "GGGCTCTGG",
                            "TTAAATAA"))
score <- c(246L, 4422L, 5679L, 1743L)
symCount <- c(10L, 6L, 9L, 8L)
axt <- Axt(targetRanges=targetRanges, targetSeqs=targetSeqs,
           queryRanges=queryRanges, querySeqs=querySeqs,
           score=score, symCount=symCount)

## getters
names(axt)
length(axt)
first(axt)
last(axt)
seqnames(axt)
strand(axt)
seqinfo(axt)

## Vector methods
axt[1]

## List methods
unlist(axt)

## Combining
c(axt, axt)
```

`axtChain` *axtChain*

Description

Wrapper function of `axtChain`: chain together psl alignments. If two matching alignments next to each other are close enough, they are joined into one segment. This function doesn't work on Windows platform since Kent utilities only support Unix-based platforms.

Usage

```
axtChain(psls, chains=sub("\\.psl$", ".chain", psls, ignore.case=TRUE),
         assemblyTarget, assemblyQuery,
         distance=c("far", "medium", "near"),
         removePsl=TRUE, binary="axtChain")
```

Arguments

`psls`	`character`(n): file names of input *psl* files.
`chains`	`character`(n): file names of output *chain* files. By default, in the same folder of input lav files with same names.
`assemblyTarget`	`character`(1): the file name of target assembly *twoBit* file.
`assemblyQuery`	`character`(1): the file name of query assembly *twoBit* file.

distance	It can be "far", "medium" or "close". It decides the score matrix used in *lastz* aligner. See '?scoringMatrix' for more details.
removePsl	boolean: When TRUE, the input *psl* files will be removed from the conversion.
binary	character(1): the name/filename of the binary axtChain to call.

Value

character(n): the file names of output *chain* files.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

lavToPsl

Examples

```
## Not run:
## This example doesn't run because it requires two bit files and external
## Kent utilities.
  psls <- tools::list_files_with_exts(
            dir="/Users/gtan/OneDrive/Project/CSC/CNEr/axt", exts="psl")
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  axtChain(psls, assemblyTarget=assemblyTarget,
           assemblyQuery=assemblyQuery, distance="far",
           removePsl=FALSE, binary="axtChain")

## End(Not run)
```

---

axtInfo *axtInfo function*

Description

Given the path of the axt file, this function retrieves information on the widths of the alignments.

Usage

```
axtInfo(axtFiles)
```

Arguments

axtFiles	The filenames of axt files.

Value

A vector of `integer` is returned. It stores the widths of all the alignments.

Author(s)

Ge Tan

See Also

`readAxt`

Examples

```
axtFile <- file.path(system.file("extdata", package="CNEr"),
                     "hg38.danRer10.net.axt")
axtInfo <- axtInfo(axtFile)
```

---

binning-utils *UCSC bin indexing system utility functions*

Description

Utility functions for UCSC bin indexing system manipulation

Usage

```
binFromCoordRange(starts, ends)
binRangesFromCoordRange(start, end)
binRestrictionString(start, end, field="bin")
```

Arguments

`starts`, `ends`	A vector of integers. A set of ranges.
`start`, `end`	A integer vector of length 1. A coordinate range.
`field`	Name of bin column. Default: "bin".

Details

The UCSC bin indexing system was initially suggested by Richard Durbin and Lincoln Stein to speed up the SELECT of a SQL query for the rows overlapping with certain genome coordinate. The system first used in UCSC genome browser is described by Kent et. al. (2002).

Value

For `binFromCoordRange`, it returns the bin number that should be assigned to a feature spanning the given range. Usually it is used when creating a database for the features.

For `binRangesFromCoordRange`, it returns the set of bin ranges that overlap a given coordinate range. It is usually used to find out the bins overlapped with a range. For SQL query, it is more convenient to use `binRestrictionString` than to use this function directly.

For `binRestrictionString`, it returns a string to be used in the WHERE section of a SQL SELECT statement that is to select features overlapping a certain range. * USE THIS WHEN QUERYING A DB *

Author(s)

Ge Tan

References

Kent, W. J., Sugnet, C. W., Furey, T. S., Roskin, K. M., Pringle, T. H., Zahler, A. M., & Haussler, A. D. (2002). The Human Genome Browser at UCSC. Genome Research, 12(6), 996-1006. doi:10.1101/gr.229102

http://genomewiki.ucsc.edu/index.php/Bin_indexing_system

Examples

```
binFromCoordRange(starts=c(10003, 1000000), ends=c(10004, 1100000))
binRangesFromCoordRange(start=10000, end=2000000)
binRestrictionString(start=10000, end=2000000, field="bin")
```

---

blatCNE *Wrapper function of blat for* CNE *object*

Description

This wrapper function blat the CNEs against the reference genome. Note that blat must be installed on your system.

Usage

```
blatCNE(cne, blatOptions=NULL, cutIdentity=90)
```

Arguments

cne	cne object after cneMerge step.
blatOptions	character(1): the blat options. When it is NULL, the options will be chosen based on the window size for scanning CNEs.
cutIdentity	integer(1): the minimum sequence identity (in percent) for a match in blat. By default, it is 90.

Details

When winSize > 45, the blat option is "-tileSize=11 -minScore=30 -repMatch=1024".

When 35 < winSize <= 45, the blat option is "-tileSize=10 -minScore=28 -repMatch=4096".

When the winSize <= 35, the blat option is "-tileSize=9 -minScore=24 -repMatch=16384".

Value

A CNE object with a final set of CNEs.

Author(s)

Ge Tan

Examples

```
## Not run:
  data(CNEDanRer10Hg38)
  data(CNEHg38DanRer10)
  cne <- CNE(assembly1Fn=file.path(system.file("extdata",
                                  package="BSgenome.Drerio.UCSC.danRer10"),
                        "single_sequences.2bit"),
             assembly2Fn=file.path(system.file("extdata",
                                  package="BSgenome.Hsapiens.UCSC.hg38"),
                        "single_sequences.2bit"),
             window=50L, identity=45L, CNE12=CNEDanRer10Hg38[["45_50"]],
             CNE21=CNEHg38DanRer10[["45_50"]], aligner="blat")
  cne <- cneMerge(cne)
  cne <- blatCNE(cne)

## End(Not run)
```

---

ceScan-methods *ceScan function*

Description

This is the main function for conserved noncoding elements (CNEs) identification.

Usage

```
ceScan(x, tFilter=NULL, qFilter=NULL,
       tSizes=NULL, qSizes=NULL, window=50L, identity=50L)
```

Arguments

x	CNE object, or Axt object, or character(n) object of Axt filenames.
tFilter, qFilter	GRanges object or NULL: regions to filter out for target and query assembly.
tSizes, qSizes	Seqinfo object or integer(n) or NULL: it contains the seqnames and seqlengths for target and query genome. When it's NULL, this 'seqinfo' must exist in 'x'.
window	integer(n): the window size of scanning CNEs. By default, it is 50L.
identity	integer(n): the minimal identity score over the scanning window. By default, it is 50L.

Details

ceScan scan the axts alignments and identify the CNEs. ceScan can accept axts in Axt object and regions to filter out as GRanges objects, or directly the 'axt' files and 'bed' files.

The details of the algorithm are described in the vignette.

Value

A list of GRangePairs or CNE object is returned. Each element of the list corresponds to one user-specified threshold for identifying CNEs.

Author(s)

Ge Tan

Examples

```
library(BSgenome.Drerio.UCSC.danRer10)
library(BSgenome.Hsapiens.UCSC.hg38)
axtFnHg38DanRer10 <- file.path(system.file("extdata", package="CNEr"),
                               "hg38.danRer10.net.axt")
axtHg38DanRer10 <- readAxt(axtFnHg38DanRer10)
axtFnDanRer10Hg38 <- file.path(system.file("extdata", package="CNEr"),
                               "danRer10.hg38.net.axt")
axtDanRer10Hg38 <- readAxt(axtFnDanRer10Hg38)
bedHg38Fn <- file.path(system.file("extdata", package="CNEr"),
                       "filter_regions.hg38.bed")
bedHg38 <- readBed(bedHg38Fn)
bedDanRer10Fn <- file.path(system.file("extdata", package="CNEr"),
                           "filter_regions.danRer10.bed")
bedDanRer10 <- readBed(bedDanRer10Fn)
qSizesHg38 <- seqinfo(BSgenome.Hsapiens.UCSC.hg38)
qSizesDanRer10 <- seqinfo(BSgenome.Drerio.UCSC.danRer10)

## Axt object
windows <- c(50L, 50L, 50L)
identities <- c(45L, 48L, 49L)
CNEHg38DanRer10 <- ceScan(x=axtHg38DanRer10, tFilter=bedHg38,
                          qFilter=bedDanRer10,
                          tSizes=qSizesHg38, qSizes=qSizesDanRer10,
                          window=windows, identity=identities)
CNEDanRer10Hg38 <- ceScan(x=axtDanRer10Hg38, tFilter=bedDanRer10,
                          qFilter=bedHg38,
                          tSizes=qSizesDanRer10, qSizes=qSizesHg38,
                          window=windows, identity=identities)

## CNE object
cneDanRer10Hg38 <- CNE(
  assembly1Fn=file.path(system.file("extdata",
                                    package="BSgenome.Drerio.UCSC.danRer10"),
                        "single_sequences.2bit"),
  assembly2Fn=file.path(system.file("extdata",
                                    package="BSgenome.Hsapiens.UCSC.hg38"),
                        "single_sequences.2bit"),
  axt12Fn=axtFnDanRer10Hg38, axt21Fn=axtFnHg38DanRer10,
  cutoffs1=8L, cutoffs2=4L)
## Here danRer10Filter is tFilter since danRer10 is assembly1
cneListDanRer10Hg38 <- ceScan(x=cneDanRer10Hg38, tFilter=bedDanRer10,
                              qFilter=bedHg38,
                              window=windows, identity=identities)
```

---

chainMergeSort *chainMergeSort*

Description

Wrapper function of `chainMergeSort`: Combines sorted files into a larger sorted file. This function doesn't work on Windows platform since Kent utilities only support Linux and Unix platforms.

Usage

```
chainMergeSort(chains, assemblyTarget, assemblyQuery,
               allChain=paste0(sub("\\.2bit$", "", basename(assemblyTarget),
                                   ignore.case=TRUE), ".",
                               sub("\\.2bit$", "", basename(assemblyQuery),
                                   ignore.case=TRUE), ".all.chain"),
               removeChains=TRUE, binary="chainMergeSort")
```

Arguments

`chains`	`character`(n): file names of input *chains* files.
`assemblyTarget`	`character`(1): the file name of target assembly *twoBit* file.
`assemblyQuery`	`character`(1): the file name of query assembly *twoBit* file.
`allChain`	`character`(1): file names of merged *allChain* file.
`removeChains`	`boolean`: When `TRUE`, the input *chains* files will be removed after the conversion.
`binary`	`character`(1): the name/filename of the binary `chainMergeSort` to call.

Details

This *allChain* file is what we get from UCSC download, *e.g.*, **hg19.danRer7.all.chain.gz**.

Value

`character`(1): the file names of merged *allChain* file.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

`axtChain`

Examples

```
## Not run:
## This example doesn't run because it requires two bit files and external
## Kent utilities.
  chains <- tools::list_files_with_exts(
              dir="/Users/gtan/OneDrive/Project/CSC/CNEr/axt", exts="chain")
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  chainMergeSort(chains, assemblyTarget, assemblyQuery,
```

```
    allChain=file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                       paste0(sub("\\.2bit$", "", basename(assemblyTarget),
                                  ignore.case=TRUE), ".",
                              sub("\\.2bit$", "", basename(assemblyQuery),
                                  ignore.case=TRUE), ".all.chain")),
    removeChains=FALSE, binary="chainMergeSort")

## End(Not run)
```

`chainNetSyntenic` *chainNetSyntenic*

Description

Wrapper function of *chainNetSyntenic*: Makes alignment nets out of chains and adds synteny info to net. This function doesn't work on Windows platform since Kent utilities only support Linux and Unix platforms.

Usage

```
chainNetSyntenic(allPreChain, assemblyTarget, assemblyQuery,
                 netSyntenicFile=paste0(sub("\\.2bit$", "",
                                            basename(assemblyTarget),
                                            ignore.case = TRUE), ".",
                                        sub("\\.2bit$", "",
                                            basename(assemblyQuery),
                                            ignore.case = TRUE),
                                            ".noClass.net"),
                 binaryChainNet="chainNet", binaryNetSyntenic="netSyntenic")
```

Arguments

`allPreChain`	`character(1)`: file names of input *allPreChain* file.
`assemblyTarget`	`character(1)`: the file name of target assembly *twoBit* file.
`assemblyQuery`	`character(1)`: the file name of query assembly *twoBit* file.
`netSyntenicFile`	`character(1)`: file names of output *netSyntenicFile* file.
`binaryChainNet`	`character(1)`: the name/filename of the binary `chainNet` to call.
`binaryNetSyntenic`	`character(1)`: the name/filename of the binary `netSyntenic` to call.

Details

Add classification information using the database tables: actually this step is not necessary in this pipeline according to http://blog.gmane.org/gmane.science.biology.ucscgenome.general/month=20130301. The class information will only be used for Genome Browser. Since it needs some specific modification of the table names for certain species, we skip this step now. If this step is done, then the generated *class.net* is the gzipped net file that you see in UCSC Downloads area.

Value

`character(1)`: the file names of generated *net* file.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

chainPreNet

Examples

```
## Not run:
## This example doesn't run because it requires two bit files and external
## Kent utilities.
  allPreChain <- file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                           "danRer10.hg38.all.pre.chain")
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  chainNetSyntenic(allPreChain, assemblyTarget, assemblyQuery,
                   netSyntenicFile=file.path(
                     "/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                                      paste0(sub("\\.2bit$", "",
                                             basename(assemblyTarget),
                                             ignore.case = TRUE), ".",
                                             sub("\\.2bit$", "",
                                             basename(assemblyQuery),
                                             ignore.case = TRUE),
                                    ".noClass.net")),
                   binaryChainNet="chainNet", binaryNetSyntenic="netSyntenic")

## End(Not run)
```

---

chainPreNet *chainPreNet*

Description

Wrapper function of chainPreNet: Removes chains that don't have a chance of being netted. This function doesn't work on Windows platform since Kent utilities only support Linux and Unix platforms.

Usage

```
chainPreNet(allChain, assemblyTarget, assemblyQuery,
            allPreChain=paste0(sub("\\.2bit$", "", basename(assemblyTarget),
                               ignore.case = TRUE), ".",
                           sub("\\.2bit$", "", basename(assemblyQuery),
                               ignore.case = TRUE), ".all.pre.chain"),
            removeAllChain=TRUE, binary="chainPreNet")
```

Arguments

allChain	character(1): file names of input *allChain* file.
assemblyTarget	character(1): the file name of target assembly *twoBit* file.
assemblyQuery	character(1): the file name of query assembly *twoBit* file.
allPreChain	character(1): file names of merged *allPreChain* file.
removeAllChain	boolean: When TRUE, the input *allChain* file will be removed after the conversion.
binary	character(1): the name/filename of the binary chainPreNet to call.

Value

character(1): the file names of merged *allPreChain* file.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

chainMergeSort

Examples

```
## Not run:
## This example doesn't run because it requires two bit files and external
## Kent utilities.
  allChain <- file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                        "danRer10.hg38.all.chain")
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  chainPreNet(allChain, assemblyTarget, assemblyQuery,
              allPreChain=file.path(
                            "/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                                    paste0(sub("\\.2bit$", "",
                                               basename(assemblyTarget),
                                               ignore.case = TRUE), ".",
                                           sub("\\.2bit$", "",
                                               basename(assemblyQuery),
                                               ignore.case = TRUE),
                                               ".all.pre.chain")),
              removeAllChain=FALSE, binary="chainPreNet")

## End(Not run)
```

CNE-class *Class* "CNE"

Description

CNE class contains all the meta-data of CNEs, including the pair of assemblies, the thresholds, the intermediate and final CNE sets.

Usage

```
### Constructors:
CNE(assembly1Fn=character(1), assembly2Fn=character(1),
    axt12Fn=character(), axt21Fn=character(),
    window=50L, identity=50L,
    CNE12=GRangePairs(), CNE21=GRangePairs(),
    CNEMerged=GRangePairs(), CNEFinal=GRangePairs(),
    aligner="blat", cutoffs1=4L, cutoffs2=4L)

### Accessor-like methods:
## S4 method for signature 'CNE'
thresholds(x)
## S4 method for signature 'CNE'
CNE12(x)
## S4 method for signature 'CNE'
CNE21(x)
## S4 method for signature 'CNE'
CNEMerged(x)
## S4 method for signature 'CNE'
CNEFinal(x)

## ... and more (see Methods)
```

Arguments

assembly1Fn, assembly2Fn	Object of class "character": The twoBit filenames of assembly1, assembly2
axt12Fn, axt21Fn	Object of class "character": The Axt filenames of assembly1 to assembly2, assembly2 to assembly1
window	Object of class "integer": The window size for scanning CNEs. By default, it is 50.
identity	Object of class "integer": The identity over the window size for scanning CNEs. By default, it is 50.
CNE12	Object of class "GRangePairs": The preliminary CNEs from axt file with assembly1 as reference.
CNE21	Object of class "GRangePairs": The preliminary CNEs from axt file with assembly2 as reference.
CNEMerged	Object of class "GRangePairs": The CNEs after merging CNE1 and CNE2.
CNEFinal	Object of class "GRangePairs": The CNEs after being realigned back to reference genome, with blat in current implementation.

`aligner` Object of class `"character"`: The method to realign CNEs back to the reference genome.

`cutoffs1`, `cutoffs2` Object of class `"integer"`: The CNEs with more than the cutoff hits on the reference genome are removed.

`x` Object of class `"CNE"`: A `"CNE"` object.

Methods

CNE12 `signature(x = "CNE")`: Get the CNE1 results.

CNE21 `signature(x = "CNE")`: Get the CNE2 results.

CNEMerged `signature(x = "CNE")`: Get the merged CNE results.

CNEFinal `signature(x = "CNE")`: Get the final CNE results.

thresholds `signature(x = "CNE")`: Get the thresholds used for scanning CNEs.

Author(s)

Ge Tan

Examples

```
library(GenomicRanges)
## Constructor
CNE12 <- GRangePairs(first=GRanges(seqnames=c("chr13", "chr4", "chr4"),
                                   ranges=IRanges(start=c(71727138,150679343,
                                                          146653164),
                                                  end=c(71727224, 150679400,
                                                        146653221)),
                                   strand="+"),
                     second=GRanges(seqnames=c("chr1"),
                                    ranges=IRanges(start=c(29854162, 23432387,
                                                           35711077),
                                                   end=c(29854248, 23432444,
                                                         35711134)),
                                    strand="+")
                     )
CNE21 <- GRangePairs(first=GRanges(seqnames=c("chr1"),
                                   ranges=IRanges(start=c(29854162, 23432387,
                                                          35711077),
                                                  end=c(29854248, 23432444,
                                                        35711134)),
                                   strand="+"),
                     second=GRanges(seqnames=c("chr13", "chr4", "chr4"),
                                    ranges=IRanges(start=c(71727138,150679343,
                                                           146653164),
                                                   end=c(71727224, 150679400,
                                                         146653221)),
                                    strand="+")
                     )
cne <- CNE(assembly1Fn=file.path(system.file("extdata",
                                   package="BSgenome.Drerio.UCSC.danRer10"),
                      "single_sequences.2bit"),
           assembly2Fn=file.path(system.file("extdata",
                                   package="BSgenome.Hsapiens.UCSC.hg38"),
```

```
                    "single_sequences.2bit"),
            window=50L, identity=50L,
            CNE12=CNE12, CNE21=CNE21, CNEMerged=CNE12, CNEFinal=CNE12,
            aligner="blat", cutoffs1=4L, cutoffs2=4L)

## Accessor
CNE12(cne)
CNE21(cne)
thresholds(cne)
CNEMerged(cne)
CNEFinal(cne)
```

`CNEDanRer10Hg38` *CNEHg38DanRer10 and CNEDanRer10Hg38 dataset*

Description

These two datasets are the direct output from `ceScan`.

Usage

```
data(CNEHg38DanRer10)
```

Examples

```
data(CNEHg38DanRer10)
```

`CNEDensity-methods` *CNEDensity function*

Description

This function queries the database and generates the CNEs' density values.

Usage

```
CNEDensity(dbName, tableName, chr, start, end,
           whichAssembly=c("first", "second"),
           windowSize=300, minLength=NULL)
```

Arguments

`dbName`	`character(1)`: the path of the local SQLite database.
`tableName`	`character(1)`: the name of table for this CNE data table. It can be missing when assembly1, assembly2 and threshold are provided.
`chr`	`character(1)`: the chromosome to query.
`start, end`	`integer(1)`: the start and end coordinate to fetch the CNEs.
`whichAssembly`	`character(1)`: the genome to fetch is in the 'first' column or 'second' column of the table.
`windowSize`	`integer(1)`: the window size in kb that is used to smooth the CNEs.
`minLength`	`integer(1)`: the minimal length of CNEs to fetch.

Value

A GRanges object with density values is returned.

Methods

signature(tableName = "character", assembly1 = "character", assembly2 = "missing", threshold = "missi

signature(tableName = "missing", assembly1 = "character", assembly2 = "character", threshold = "chara

Author(s)

Ge Tan

Examples

```
dbName <- file.path(system.file("extdata", package="CNEr"),
                    "danRer10CNE.sqlite")
genome <- "danRer10"
chr <- "chr6"
start <- 24000000L
end <-  27000000L
windowSize <- 200L
minLength <- 50L
cneDanRer10Hg38_45_50 <-
  CNEDensity(dbName=dbName,
             tableName="danRer10_hg38_45_50",
             whichAssembly="first", chr=chr, start=start,
             end=end, windowSize=windowSize,
             minLength=minLength)
cneDanRer10Hg38_49_50 <-
  CNEDensity(dbName=dbName,
             tableName="danRer10_hg38_49_50",
             whichAssembly="first", chr=chr, start=start,
             end=end, windowSize=windowSize,
             minLength=minLength)
```

---

cneFinalListDanRer10Hg38

cneFinalListDanRer10Hg38 dataset

Description

cneFinalListDanRer10Hg38 dataset contains the CNE between danRer10 and hg38 around chr6:24,000,000..27,000,000.

Usage

```
data("cneFinalListDanRer10Hg38")
```

Examples

```
data(cneFinalListDanRer10Hg38)
```

cneMerge-methods *CNE merge function*

Description

Removes the CNEs which overlap on both genomes.

Usage

```
cneMerge(cne12, cne21)
```

Arguments

cne12	A object of CNE or GRangePairs.
cne21	A object of GRangePairs object. When cne12 is a CNE object, cne21 can be missing.

Value

A GRangePairs of CNEs or a CNE object is returned. In this table, the order of columns is consistent with cne1. For instance, if cne1 has the first three columns for zebrafish and next three columns for human, in the merged table, the first three columns are still the coordinates for zebrafish while the next three columns are the coordinates for human.

Author(s)

Ge Tan

Examples

```
library(GenomicRanges)
firstGRange <- GRanges(seqnames=c("chr1", "chr1", "chr2", "chr2", "chr5"),
                       ranges=IRanges(start=c(1, 20, 2, 3, 1),
                                      end=c(10, 25, 10, 10, 10)),
                       strand="+")
lastGRange <- GRanges(seqnames=c("chr15", "chr10", "chr10", "chr10", "chr15"),
                      ranges=IRanges(start=c(1, 25, 50, 51, 5),
                                     end=c(8, 40, 55, 60, 10)),
                      strand="+")
cne12 <- GRangePairs(firstGRange[1:3], lastGRange[1:3])
cne21 <- GRangePairs(lastGRange[4:5], firstGRange[4:5])

## GRangePairs, GRangePairs
cneMerge(cne12, cne21)

## CNE, missing
cne <- CNE(assembly1Fn=file.path(system.file("extdata",
                                 package="BSgenome.Drerio.UCSC.danRer10"),
                      "single_sequences.2bit"),
           assembly2Fn=file.path(system.file("extdata",
                                 package="BSgenome.Hsapiens.UCSC.hg38"),
                      "single_sequences.2bit"),
           window=50L, identity=50L,
```

```
            CNE12=cne12, CNE21=cne21, aligner="blat")
cneMerge(cne)
```

`fetchChromSizes` *fetchChromSizes function.*

Description

This function tries to automate the fetch of chromosome sizes for assemblies from UCSC.

Usage

```
fetchChromSizes(assembly)
```

Arguments

`assembly`	A `character` object: the canonical name of assembly, i.e., 'hg19' for UCSC.

Details

This function downloads 'chromInfo.txt.gz' from UCSC golden path for UCSC assemblies.

Value

A object of `Seqinfo` is returned.

Note

Currently, the assemblies from UCSC are supported.

Author(s)

Ge Tan

Examples

```
fetchChromSizes("hg19")
fetchChromSizes("mm10")
```

`fixCoordinates` *Fix the coordinates in* `Axt` *object*

Description

In 'axt' file and `Axt` object, the coordinates of negative query alignments are relative to the reverse-complemented coordinates of its chromosome. This is different from the convention in Bioconductor. This function fixes the coordinates which are always relative to the positive strand.

Usage

```
fixCoordinates(x)
```

Arguments

`x`	`Axt` object.

Details

In `Axt`, the 'strand' is for the aligning organism. If the strand value is "-", the values of the aligning organism's start and end fields are relative to the reverse-complemented coordinates of its chromosome.

Value

A `Axt` object.

Author(s)

Ge Tan

Examples

```
axtFnDanRer10Hg38 <- file.path(system.file("extdata", package="CNEr"),
                               "danRer10.hg38.net.axt")
qAssemblyFn <- file.path(system.file("extdata",
                                     package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
tAssemblyFn <- file.path(system.file("extdata",
                                     package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
axtDanRer10Hg38 <- readAxt(axtFnDanRer10Hg38, tAssemblyFn=tAssemblyFn,
                           qAssemblyFn=qAssemblyFn)
## Fix the coordinates
fixCoordinates(axtDanRer10Hg38)

## Restore it
fixCoordinates(fixCoordinates(axtDanRer10Hg38))
```

GRangePairs-class *GRangePairs objects*

Description

The GRangePairs class is a container for a pair of GRanges objects that have the same lengths.

Details

A GRangePairs object is a list-like object where each element describes a pair of genomic range. They do not necessarily have the same seqinfo, *i.e.*, the coordinates from the same assembly.

Constructor

GRangePairs(first=GRanges(), second=GRanges(), ..., names=NULL, hits=NULL): GRangePairs constructor.

Accessors

In the code snippets below, x is a GRangePairs object.

length(x): Return the number of granges pairs in x.

names(x), names(x) <- value: Get or set the names on x.

first(x), last(x), second(x): Get the 'first' or 'last'/'second' GRange for each grange pair in x. The result is a GRanges object of the same length as x.

first(x)<-, second(x)<-: Set the 'first' or 'second' GRange for each grange pair in x. The result is a GRanges object of the same length as x.

seqnames(x): Get the seqname of first GRanges and last GRanges and return in a DataFrame object.

strand(x): Get the strand for each grange pair in x.

seqinfo(x): Get the information about the underlying sequences.

Vector methods

In the code snippets below, x is a GRangePairs object.

x[i]: Return a new GRangePairs object made of the selected genomic ranges pairs.

List methods

In the code snippets below, x is a GRangePairs object.

unlist(x, use.names=TRUE): Return the GRangePairs object conceptually defined by c(x[[1]], x[[2]], ..., x[[length(x)]]). use.names determines whether x names should be passed to the result or not.

Coercion

In the code snippets below, x is a GRangePairs object.

`grglist(x, use.mcols=FALSE)`:
Return a GRangesList object of length `length(x)` where the i-th element represents the ranges (with respect to the reference) of the i-th grange pair in `x`.

Note that this results in the ranges being *always* ordered consistently with the original "query template", that is, being in the order defined by walking the "query template" from the beginning to the end.

If `use.mcols` is TRUE and `x` has metadata columns on it (accessible with `mcols(x)`), they're propagated to the returned object.

`as(x, "GRangesList")`: Alternate ways of doing `grglist(x, use.mcols=TRUE)`.

`as(x, "GRanges")`: Equivalent of `unlist(x, use.names=TRUE)`.

Other methods

In the code snippets below, x is a GRangesList object.

`swap(x)`: Swap the first, last GRanges.

`unique(x)`: Get the unique GRangePairs.

`show(x)`: By default, the show method displays 5 head and 5 tail elements. This can be changed by setting the global options `showHeadLines` and `showTailLines`. If the object length is less than (or equal to) the sum of these 2 options plus 1, then the full object is displayed.

Author(s)

Ge Tan

See Also

Axt

Examples

```
## Constructor
library(GenomicRanges)
first <- GRanges(seqnames=c("chr1", "chr1", "chr2", "chr3"),
                 ranges=IRanges(start=c(1, 20, 2, 3),
                                end=c(10, 25, 10, 10)),
                 strand="+")
last <- GRanges(seqnames=c("chr1", "chr10", "chr10", "chr20"),
                ranges=IRanges(start=c(1, 25, 50, 5),
                               end=c(8, 40, 55, 16)),
                strand="+")
namesGRangePairs <- c("a","b","c","d")
grangesPairs1 <- GRangePairs(first, last, names=namesGRangePairs)
grangesPairs2 <- GRangePairs(first, last)

## getters and setters
names(grangesPairs1)
names(grangesPairs2) <- namesGRangePairs

first(grangesPairs1)
```

```
first(grangesPairs1) <- second(grangesPairs1)

second(grangesPairs1)
second(grangesPairs1) <- first(grangesPairs1)

length(grangesPairs1)
seqnames(grangesPairs1)
strand(grangesPairs1)
seqinfo(grangesPairs1)

## Vector methods
grangesPairs1[1]

## List methods
unlist(grangesPairs1)

## Coersion
grglist(grangesPairs1)
as(grangesPairs1, "GRangesList")
as(grangesPairs1, "GRanges")
as(grangesPairs1, "DataFrame")
as.data.frame(grangesPairs1)

## Combining
c(grangesPairs1, grangesPairs2)

## Swap
swap(grangesPairs1)

## Unique
unique(c(grangesPairs1, grangesPairs1))
```

`grangesPairsForDotplot` *grangesPairsForDotplot*

Description

Example of `GrangePairs` object from the collinear regions of *Adineta vaga*.

Usage

```
data("grangesPairsForDotplot")
```

Details

The collinear regions from "scaffold_1" and "scaffold_5".

Source

Example from own project.

Examples

```
data(grangesPairsForDotplot)
```

`lastal` *lastal wrapper*

Description

Wrapper function of `lastal` to do the pairwise whole genome alignment. This function doesn't work on Windows platform.

Usage

```
lastal(db, queryFn,
       outputFn=sub("\\.(fa|fasta)$", ".maf",
                    paste(basename(db), basename(queryFn), sep = ","),
                    ignore.case = TRUE),
       distance=c("far", "medium", "near"), binary="lastal",
       mc.cores=getOption("mc.cores", 2L), echoCommand=FALSE)
```

Arguments

`db`	`character(1)`: the file name of target assembly's lastal index.
`queryFn`	`character(1)`: the file name of query assembly *fasta* file.
`outputFn`	`character(1)`: the file name of the output *maf* file.
`distance`	It can be "far", "medium" or "near". It decides the score matrix used in *lastz* aligner. See '?scoringMatrix' for more details.
`binary`	`character(1)`: the name/filename of the binary `lastal` to call.
`mc.cores`	`integer(1)`: the number of threads to use. By default, `getOption("mc.cores", 2L)`.
`echoCommand`	`boolean(1)`: When `TRUE`, only the command to run `lastal` is returned.

Value

A `character(1)` vector of ouput *maf* file names.

Note

`lastal` aligner must be installed on the machine to use this function.

Author(s)

Ge Tan

References

http://last.cbrc.jp/

See Also

`lastz`

Examples

```
## Not run:
  assemblyDir <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit"
  ## Build the lastdb index
  system2(command="lastdb", args=c("-c", file.path(assemblyDir, "danRer10"),
                                   file.path(assemblyDir, "danRer10.fa")))

  ## Run lastal aligner
  lastal(db=file.path(assemblyDir, "danRer10"),
       queryFn=file.path(assemblyDir, "hg38.fa"),
       outputFn=file.path(axtDir, "danRer10.hg38.maf"),
       distance="far", binary="lastal", mc.cores=4L)

  ## maf to psl
  psls <- file.path(axtDir, "danRer10.hg38.psl")
  system2(command="maf-convert",
          args=c("psl", file.path(axtDir, "danRer10.hg38.maf"),
                 ">", psls))

## End(Not run)
```

`lastz` *lastz wrapper*

Description

Wrapper function of `lastz` to do the pairwise whole genome alignment. This function doesn't work on Windows platform.

Usage

```
lastz(assemblyTarget, assemblyQuery, outputDir = ".",
      chrsTarget = NULL, chrsQuery = NULL,
      distance = c("far", "medium", "near"), binary = "lastz",
      mc.cores = getOption("mc.cores", 2L), echoCommand = FALSE)
```

Arguments

`assemblyTarget`	`character(1)`: the file name of target assembly *twoBit* file.
`assemblyQuery`	`character(1)`: the file name of query assembly *twoBit* file.
`outputDir`	`character(1)`: the folder to put the generated *lav* files.
`chrsTarget`	NULL or `character(n)`: when it's NULL, all the available chromosomes from the target assembly will be aligned.
`chrsQuery`	NULL or `character(n)`: when it's NULL, all the available chromosomes from the query assembly will be aligned.
`distance`	It can be "far", "medium" or "near". It decides the score matrix used in *lastz* aligner. See '?scoringMatrix' for more details.
`binary`	`character(1)`: the name/filename of the binary `lastz` to call.
`mc.cores`	`integer(1)`: the number of threads to use. By default, `getOption("mc.cores", 2L)`.
`echoCommand`	`boolean(1)`: When `TRUE`, only the command to run `lastz` is returned.

Value

A character(n) vector of ouput *lav* file names.

Note

lastz aligner must be installed on the machine to use this function.

Author(s)

Ge Tan

References

http://www.bx.psu.edu/~rsharris/lastz/

See Also

lavToPsl

Examples

```
## Not run:
## This example doesn't run because it requires two bit files and external
## Kent utilities.
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  lavs <- lastz(assemblyTarget, assemblyQuery,
                outputDir="/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
              chrsTarget=c("chr1", "chr2", "chr3"),
              chrsQuery=c("chr1", "chr2", "chr3"),
              distance="far", mc.cores=4)

## End(Not run)
```

---

lavToPsl *lavToPsl*

Description

Wrapper function of lavToPsl: Convert blastz lav to psl format. This function doesn't work on Windows platform since Kent utilities only support Linux and Unix platforms.

Usage

```
lavToPsl(lavs, psls=sub("\\.lav$", ".psl", lavs, ignore.case = TRUE),
         removeLav=TRUE, binary="lavToPsl")
```

Arguments

lavs	character(n): file names of input *lav* files.
psls	codecharacter(n): file names of output *psl* files. By default, in the same folder of input lav files with same names.
removeLav	boolean: When TRUE, the input *lavs* files will be removed after the conversion.
binary	character(1): the name/filename of the binary lavToPsl to call.

Value

character(n): the file names of output *psl* files.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

lastz

Examples

```
## Not run:
## This example doesn't run because it requires lav files from previous steps
## and external Kent utilities.
  lavs <- tools::list_files_with_exts(
            dir="/Users/gtan/OneDrive/Project/CSC/CNEr/axt", exts="lav")
  lavToPsl(lavs, removeLav=FALSE, binary="lavToPsl")

## End(Not run)
```

---

makeAncoraFiles *makeAncoraFiles*

Description

Make ancora format files from GRangePairs of CNE

Usage

```
makeAncoraFiles(cne, outputDir = ".",
                genomeFirst = "first", genomeSecond = "second",
                threshold = "50_50")
```

Arguments

cne	GRangePairs object of CNE.
outputDir	character(1): the output directory of 'Bed' and 'BigWig' files.
genomeFirst, genomeSecond	character(1): the genome name of the first and second species.
threshold	character(1): the threshold used to identify the CNEs in the format of "50_50" etc

Value

The filenames of output.

Note

This function is mainly for internal use in Lenhard group.

Author(s)

Ge Tan

See Also

readAncora

Examples

```
data(cneFinalListDanRer10Hg38)
cne <- CNEFinal(cneFinalListDanRer10Hg38[["45_50"]])
makeAncoraFiles(cne, genomeFirst = "danRer10", genomeSecond = "hg38",
                threshold = "45_50")
```

---

makeAxtTracks *makeAxtTracks*

Description

Make the bed tracks for the 'Axt' alignment.

Usage

```
makeAxtTracks(x)
```

Arguments

x	A Axt object.

Details

The coordinates of query 'Axt' alignment are fixed to be relative to positive strand before output into 'bed' file.

Value

A list of GRanges for target and query alignments. The two output 'bed' files are "targetAxt.bed" and "queryAxt.bed".

Author(s)

Ge Tan

See Also

fixCoordinates

Examples

```
tAssemblyFn <- file.path(system.file("extdata",
                                     package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
qAssemblyFn <- file.path(system.file("extdata",
                                     package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
axtFn <- file.path(system.file("extdata", package="CNEr"),
                   "danRer10.hg38.net.axt")
axt <- readAxt(axtFn, tAssemblyFn, qAssemblyFn)
makeAxtTracks(axt)
```

`makeCNEDensity` *Make 'Bed', 'bedGraph' and 'BigWig' files*

Description

Make 'Bed', 'bedGraph', 'BigWig' files from `GRangePairs` for display in other Genome Browser.

Usage

```
makeCNEDensity(x, outputDir = ".",
               genomeFirst = "first", genomeSecond = "second",
               threshold = "50_50",
               windowSizeFirst = 300L, windowSizeSecond = 300L)
```

Arguments

`x`	`GRangePairs` object of CNEs.
`outputDir`	`character`(1): the output directory of 'Bed', 'bedGraph' and 'BigWig' files.
`genomeFirst`, `genomeSecond`	`character`(1): the genome name of the first and second species.
`threshold`	`character`(1): the threshold used to identify the CNEs in format of "50_50".
`windowSizeFirst`, `windowSizeSecond`	`integer`(1): the smoothing window size for generating the CNE density in kb.

Details

The CNE density is defined as the percentage of regions covered by CNEs within the smoothing window.

Value

The filenames of output 'Bed', 'bedGraph' and 'BigWig' files.

Note

This function is mainly for internal use in Lenhard group.

Author(s)

Ge Tan

See Also

readAncora

Examples

```
## Not run:
dbName <- file.path(system.file("extdata", package="CNEr"),
                    "danRer10CNE.sqlite")
qAssemblyFn <- file.path(system.file("extdata",
                                     package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
tAssemblyFn <- file.path(system.file("extdata",
                           package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
cneGRangePairs <- readCNERangesFromSQLite(dbName=dbName,
                                          tableName="danRer10_hg38_45_50",
                                          tAssemblyFn=tAssemblyFn,
                                          qAssemblyFn=qAssemblyFn)
makeCNEDensity(cneGRangePairs[1:1000])

## End(Not run)
```

---

makeGRBs *makeGRBs*

Description

Make Genomic Regulatory Blocks (GRBs) boundaries prediction from a set of CNEs.

Usage

```
makeGRBs(x, winSize=NULL, genes=NULL, ratio=1,
         background=c("chromosome", "genome"), minCNEs=1L)
```

Arguments

x	GRangesList object of a set of CNEs to use.
winSize	integer: the smoothing window size for CNE densities in kb. This value depends on the genome size of the reference genome. A larger genome requires bigger window size. For instance, 300kb is the appropriate window size for the human genome. By default, it is determined internally based on the genome size.
genes	NULL or GRanges object: the protein-coding genes ranges.
ratio	numeric(1): the threshold to control the stringency of the GRBs. Higher value, shorter and fewer GRBs, and vice versa.
background	character(1): can be "chromosome" or "genome". When using slice for the CNE density, the background is calculated on a per-chromosome or whole-genome basis.
minCNEs	integer(1): the minimal number of CNEs that a GRB needs to have.

Details

First we calculate the CNE densities from the CNEs. Then we segment the regions according to the values of CNE densities. The regions with CNE densities above the expected CNE densities * ratio are considered as putative GRBs. Putative GRBs that do not encompass any gene are filtered out. Finally, the GRBs that have fewer than `minCNEs` number of CNEs will be filtered out.

Value

A `GRanges` object of GRB coordinates is returned. The numbers of CNEs and the coordinates of CNEs within each GRB are returned as a metadata column.

Author(s)

Ge Tan

Examples

```
library(TxDb.Drerio.UCSC.danRer10.refGene)
refGenesDanRer10 <- genes(TxDb.Drerio.UCSC.danRer10.refGene)
ancoraCNEsFns <- file.path(system.file("extdata", package="CNEr"),
                           c("cne2wBf_cypCar1_danRer10_100_100",
                             "cne2wBf_cteIde1_danRer10_100_100",
                             "cne2wBf_AstMex102_danRer10_48_50"))
cneList <- do.call(GRangesList,
                   lapply(ancoraCNEsFns, readAncora, assembly="danRer10"))
names(cneList) <- c("Common carp", "Grass carp", "Blind cave fish")
seqlengths(cneList) <- seqlengths(TxDb.Drerio.UCSC.danRer10.refGene)[
                         names(seqlengths(cneList))]
makeGRBs(cneList, winSize=200, genes=refGenesDanRer10, ratio=1.2,
         background="genome")
makeGRBs(cneList, winSize=200, genes=refGenesDanRer10, ratio=1.2,
         background="chromosome", minCNEs=3L)
```

---

matchDistribution *Plot the distribution of matched alignments.*

Description

Given a `Axt` alignment, plot a heatmap showing the percentage of each matched alignments.

Usage

```
matchDistribution(x, size=10000, title=NULL)
```

Arguments

`x`	`Axt` object.
`size`	`integer(1)`: the number of alignments to use. By default, it is 10000.
`title`	`character(1)`: the customised title for the plot.

Details

By default, if there are more than 10,000 alignments, 10,000 alignments will be sampled and calculated for the distribution for speed purposes.

Only the four bases (A, C, G, T), gap (-) and any (N) are displayed. Other ambiguous bases are not considered.

Value

A ggplot2 object will be returned.

Author(s)

Ge Tan

Examples

```
axtFile <- file.path(system.file("extdata", package="CNEr"),
                     "hg38.danRer10.net.axt")
axt <- readAxt(axtFile)
matchDistribution(axt)
```

N50 *Assembly statistics.*

Description

Calculate the N50, N90 values for a fasta or 2bit file.

Usage

```
N50(fn)
N90(fn)
```

Arguments

fn	character(1): The path of a fasta or 2bit file.

Details

This function calculates the N50, N90 values for an assembly. The N50 value is calculated by first ordering every contig/scaffold by length from longest to shortest. Next, starting from the longest contig/scaffold, the lengths of each contig are summed, until this running sum equals one-half of the total length of all contigs/scaffolds in the assembly. Then the length of shortest contig/scaffold in this list is the N50 value. Similar procedure is used for N90 but including 90% of the assembly.

Value

An integer value of N50 or N90 value.

Author(s)

Ge Tan

Examples

```
twoBitFn <- file.path(system.file("extdata",
                                  package="BSgenome.Drerio.UCSC.danRer10"),
                      "single_sequences.2bit")
N50(twoBitFn)
```

`netToAxt` *netToAxt*

Description

Wrapper function of `netToAxt` and `axtSort`: convert net (and chain) to axt, and sort axt files. This function doesn't work on the Windows platform since Kent utilities only support Linux and Unix platforms.

Usage

```
netToAxt(in.net, in.chain, assemblyTarget, assemblyQuery,
         axtFile=paste0(sub("\\.2bit$", "", basename(assemblyTarget),
                            ignore.case = TRUE), ".",
                        sub("\\.2bit$", "", basename(assemblyQuery),
                            ignore.case = TRUE), ".net.axt"),
         removeFiles=FALSE,
         binaryNetToAxt="netToAxt", binaryAxtSort="axtSort")
```

Arguments

`in.net`	`character(1)`: file names of input *net* file.
`in.chain`	`character(1)`: file names of input *chain* file.
`assemblyTarget`	`character(1)`: the file name of target assembly *twoBit* file.
`assemblyQuery`	`character(1)`: the file name of query assembly *twoBit* file.
`axtFile`	`character(1)`: file names of output *axt* file.
`removeFiles`	`boolean`: When `TRUE`, the input *net* and *chain* files will be removed after the conversion.
`binaryNetToAxt`	`character(1)`: the name/filename of the binary `netToAxt` to call.
`binaryAxtSort`	`character(1)`: the name/filename of the binary `axtSort` to call.

Value

`character(1)`: the file name of output *axt* file.

Author(s)

Ge Tan

References

http://hgdownload.cse.ucsc.edu/admin/exe/

See Also

chainNetSyntenic

Examples

```
## Not run:
  ## This example doesn't run because it requires two bit files and external
  ## Kent utilities.
  in.net <- file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                      "danRer10.hg38.noClass.net")
  in.chain <- file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                        "danRer10.hg38.all.pre.chain")
  assemblyTarget <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/danRer10.2bit"
  assemblyQuery <- "/Users/gtan/OneDrive/Project/CSC/CNEr/2bit/hg38.2bit"
  netToAxt(in.net, in.chain, assemblyTarget, assemblyQuery,
           axtFile=file.path("/Users/gtan/OneDrive/Project/CSC/CNEr/axt",
                             paste0(sub("\\.2bit$", "",
                                        basename(assemblyTarget),
                                        ignore.case = TRUE), ".",
                                    sub("\\.2bit$", "",
                                        basename(assemblyQuery),
                                        ignore.case = TRUE),
                                    ".net.axt")),
           removeFiles=FALSE,
           binaryNetToAxt="netToAxt", binaryAxtSort="axtSort")

## End(Not run)
```

---

orgKEGGIds2EntrezIDs *Fetch mapping from KEGG IDs to Entrez IDs*

Description

Given the desired organism name, fetch the mapping between KEGG IDs and Entrez gene IDs.

Usage

```
orgKEGGIds2EntrezIDs(organism="Homo sapiens")
```

Arguments

organism	character(1): the name of organism to query. It has to be available at http://rest.kegg.jp/list/organism

Value

A list of Entrez gene IDs with KEGG IDs as names.

Author(s)

Ge Tan

Examples

```
## Not run:
  orgKEGGIds2EntrezIDs(organism="Homo sapiens")

## End(Not run)
```

`plotCNEDistribution` *Plot sequential CNE number against CNE genomic location*

Description

Plot the CNE genomic location distribution. It gives an overview of the tendency of CNEs to form clusters.

Usage

```
plotCNEDistribution(x, chrs = NULL, chrScale = c("Mb", "Kb"))
```

Arguments

`x`	`GRanges` object: the CNE locations.
`chrs`	`character`(n): the chromosomes to show. By default, the largest 6 chromosomes/scaffolds are selected.
`chrScale`	`character`(1): the chromosome/scaffold scale of 'Mb' or 'Kb' in the plot.

Details

In the plot, x axis is the genomic location along each chromosome/scaffold. The y axis is the sequential CNE number. A typical CNE cluster can be spotted by the dramatic increase in y axis and small increase in x axis.

Value

A `ggplot` object.

Author(s)

Ge Tan

See Also

`plotCNEWidth`

Examples

```
dbName <- file.path(system.file("extdata", package="CNEr"),
                    "danRer10CNE.sqlite")
qAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
tAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
cneGRangePairs <- readCNERangesFromSQLite(dbName=dbName,
                                          tableName="danRer10_hg38_45_50",
                                          tAssemblyFn=tAssemblyFn,
                                          qAssemblyFn=qAssemblyFn)
plotCNEDistribution(first(cneGRangePairs))
```

---

plotCNEWidth *Plot the CNE widths distribution*

Description

CNE widths can follow heavy tailed distribution that are associated with power-laws. This function plots the reverse cumulative density distribution of CNE widths, and fits a discrete power-law distribution. Goodness of fit can also be evaluated.

Usage

```
plotCNEWidth(x, ...)
```

Arguments

x	GRangePairs object: a pair of CNEs.
...	Additional points passed to plot function.

Details

The power-law distribution is associated with heavy tailed distribution.

A reverse cumulative density distribution plot will be generated with optimal lower bound *xmin*, scaling parameter*alpha* for power-law fit.

Value

An invisible list of fitted model is returned.

Note

The power-law distribution implementation is based on the **poweRlaw** package.

Author(s)

Ge Tan

References

Salerno, W., Havlak, P., and Miller, J. (2006). Scale-invariant structure of strongly conserved sequence in genomic intersections and alignments. Proc. Natl. Acad. Sci. U.S.A. 103, 13121-13125.

Examples

```
dbName <- file.path(system.file("extdata", package="CNEr"),
                    "danRer10CNE.sqlite")
cneGRangePairs <- readCNERangesFromSQLite(dbName=dbName,
                                          tableName="danRer10_hg38_45_50")
plotCNEWidth(cneGRangePairs)
```

---

psubAxt *Parallel subset of Axt alignment*

Description

Given two GRanges objects, select the Axt alignments whose the target and query alignments are both within each pair of ranges.

Usage

```
psubAxt(x, targetSearch, querySearch)
```

Arguments

x	Axt object.
targetSearch, querySearch	GRanges objects: the ranges to keep for target and query alignments. They must be of the same length. Strand information is ignored.

Details

The 'targetSearch' and 'querySearch' have the coordinates relative to the positive strand. For each pair of the ranges, the alignments that lie within both the target and query range are kept.

Value

A Axt object.

Author(s)

Ge Tan

See Also

psubAxt

Examples

```
library(GenomicRanges)
tAssemblyFn <- file.path(system.file("extdata",
                          package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
qAssemblyFn <- file.path(system.file("extdata",
                          package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
axtFn <- file.path(system.file("extdata", package="CNEr"),
                   "danRer10.hg38.net.axt")
axt <- readAxt(axtFn, tAssemblyFn, qAssemblyFn)

targetSearch <- GRanges(seqnames=c("chr6"),
                        ranges=IRanges(start=c(24000000, 26900000),
                                       end=c(24060000, 26905000)),
                        strand="+"
                        )
querySearch <- GRanges(seqnames=c("chr7", "chr2"),
                       ranges=IRanges(start=c(12577000, 241262700),
                                      end=c(12579000, 241268600)),
                       strand="+"
                       )
psubAxt(axt, targetSearch, querySearch)
```

`queryCNEData` *Query the CNEData package to fetch the CNEs*

Description

Query the CNEData package to fetch the CNEs based on target, query species, winSize and identity.

Usage

```
queryCNEData(dbName, target, query, winSize, identity,
             type=c("target", "all"))
```

Arguments

`dbName`	The path of SQLite database.
`target`, `query`	The CNEs between target and query species.
`winSize`, `identity`	The thresholds of CNEs to fetch on identity over winSize.
`type`	Which set of CNEs are returned. When it is "all", the CNEs of target always on the left side of returned data.frame.

Value

A data.frame of CNEs coordinates in chr, start, end.

Author(s)

Ge Tan

read.rmMask.GRanges *Read a RepeatMasker .out file*

Description

Read a RepeatMasker .out file into a GRanges object.

Usage

```
read.rmMask.GRanges(fn)
```

Arguments

fn	character(1): the filename of a RepeatMasker .out file.

Value

A GRanges object with metadata columns containing the name of the matching interspersed repeat, the class of the repeat and the Smith-Waterman score of the match.

Author(s)

Ge Tan

References

http://www.repeatmasker.org/webrepeatmaskerhelp.html

Examples

```
fn <- system.file("extdata", "ce2chrM.fa.out", package="IRanges")
read.rmMask.GRanges(fn)
```

read.rmskFasta *Read a soft repeat masked fasta*

Description

Read a soft repeat masked fasta file into a GRanges object.

Usage

```
read.rmskFasta(fn)
```

Arguments

fn	character(1): The filename of the soft repeat masked fasta.

Details

Only the lower case based ("a", "c", "g", "t") are considered in the soft repeat masked fasta.

Value

GRanges object with coordinates of repeat masked regions.

Author(s)

Ge Tan

See Also

read.rmMask.GRanges

Examples

```
fn <- file.path(system.file("extdata", package="CNEr"),
                "rmsk.fa")
read.rmskFasta(fn)
```

---

readAncora *Read the cne file from Ancora format.*

Description

Read the Ancora CNE file into a GRanges or GRangePairs object.

Usage

```
readAncora(fn, assembly=NULL, tAssemblyFn=NULL, qAssemblyFn=NULL)
```

Arguments

fn	character(1): the path of the Ancora CNE file in the format of "cne2wBf_hg38_mm10_50_50".
assembly	character(1): the assembly to fetch. When it is NULL, CNEs on both assemblies are returned.
tAssemblyFn, qAssemblyFn	character(1): filename of the 'twoBit' or 'fasta' file for the target and query genomes.

Details

The Ancora CNE filename has its own naming style. For example, "cne2wBf_hg38_mm10_50_50" denotes human coordinates for the first three columns of the file and mouse coordinates from the forth to the sixth column.

The start coordinate system is 0-based.

Value

A GRanges object of the CNE ranges when assembly is specified, or a GRangePairs object when assembly is NULL.

Note

This function is mainly for internal use in Lenhard group.

Author(s)

Ge Tan

Examples

```
fn <- file.path(system.file("extdata", package="CNEr"),
                "cne2wBf_danRer10_hg38_45_50")
zebrafishCNEs <- readAncora(fn, "danRer10")
humanCNEs <- readAncora(fn, "hg38")
zebrafishHumanCNEs <- readAncora(fn)
```

---

readAncoraIntoSQLite *Read Ancora legacy CNE format*

Description

Read Ancora legacy CNE format into a SQLite database.

Usage

```
readAncoraIntoSQLite(cneFns, dbName, overwrite=FALSE)
```

Arguments

cneFns	character(n): filenames of Ancora CNE files.
dbName	character(1): filename of SQLite database.
overwrite	boolean(1): whether or not to overwrite the existing table.

Details

The Ancora legacy CNE file has the filename in the format of "cne2wBf_AstMex102_danRer10_48_50". The first six columns are the coordinates of pairs of CNEs. The start coordinate system is 0-based and is converted into 1-based when it is imported into the SQLite database.

Value

A character vector of table names.

Note

This function is mainly for internal use in Lenhard group.

Author(s)

Ge Tan

See Also

readAncora

Examples

```
ancoraCNEsFns <- file.path(system.file("extdata", package="CNEr"),
                           c("cne2wBf_cypCar1_danRer10_100_100",
                             "cne2wBf_cteIde1_danRer10_100_100",
                             "cne2wBf_AstMex102_danRer10_48_50"))
dbName <- tempfile()
readAncoraIntoSQLite(ancoraCNEsFns, dbName, overwrite=FALSE)
```

`readAxt` *Read 'Axt' file*

Description

This function reads the 'Axt' files into an `Axt` object.

Usage

```
readAxt(axtFiles, tAssemblyFn=NULL, qAssemblyFn=NULL)
```

Arguments

`axtFiles`	`character`(n): filenames of the 'Axt' files to read.
`tAssemblyFn, qAssemblyFn`	`character`(1): filename of the 'twoBit' or 'fasta' file for the target and query genome.

Details

This function reads the 'Axt' files of two assemblies. It can be a single big 'Axt' file or several small 'Axt' files. Contrary to the start coordinate in 'Axt' file, the start coordinate in `Axt` object is 1-based.

When 'tAssemblyFn' and 'qAssemblyFn' are not `NULL`, the corresponding `Seqinfo` will be added into the returned `Axt` object.

Value

A object `Axt` is returned.

Author(s)

Ge Tan

See Also

`Axt`

Examples

```
axtFile <- file.path(system.file("extdata", package="CNEr"),
                     "hg38.danRer10.net.axt")
tAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
qAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
axt <- readAxt(axtFile, tAssemblyFn, qAssemblyFn)
```

`readBed` *Read bed file*

Description

Read the coordinates information from a bed file.

Usage

```
readBed(bedFile, assemblyFn=NULL)
```

Arguments

`bedFile`	`character(1)`: filename of the 'bed' file to read.
`assemblyFn`	`character(1)`: filename of the twoBit or fasta file of the genome.

Details

This function is designed to read the bed file as 'chrom', 'chromStart', 'chromEnd'. The strand information is also stored where available.

In the bed file, the 'chromStart' is on the 0-based coordinate system while 'chromEnd' is on the 1-based coordinate system. For example, the first 100 bases of a chromosome are defined as 'chromStart=0', 'chromEnd=100', and span the bases numbered 0-99. When it is read into `GRanges`, both the 'chromStart' and 'chromEnd' are on 1-based coordinate, *i.e.*, 'chromStart=1' and 'chromEnd=100'.

When 'assemblyFn' is not `NULL`, the corresponding `Seqinfo` will be added into the returned `GRanges`.

Value

A `GRanges` object is returned. When no strand information is available in the bed file, all the ranges are assumed to be on the positive strand.

Author(s)

Ge Tan

References

https://genome.ucsc.edu/FAQ/FAQformat.html#format1

See Also

import.bed

Examples

```
bedFn <- file.path(system.file("extdata", package="CNEr"),
                   "filter_regions.hg38.bed")
assemblyFn <- file.path(system.file("extdata",
                        package="BSgenome.Hsapiens.UCSC.hg38"),
                        "single_sequences.2bit")
bed <- readBed(bedFn, assemblyFn=assemblyFn)
```

---

readCNERangesFromSQLite

readCNERangesFromSQLite function

Description

Query the SQLite database based on chromosome, coordinates and some other criteria. Primarily not intended to be used directly. For the CNE density plot, `fetchCNEDensity` function should be used.

Usage

```
readCNERangesFromSQLite(dbName, tableName, chr=NULL, start=NULL, end=NULL,
                        whichAssembly=c("first","second"), minLength=NULL,
                        tAssemblyFn=NULL, qAssemblyFn=NULL)
```

Arguments

`dbName`	A object of `character`, the path of the local SQLite database.
`tableName`	A object of `character`, the name of table for this CNE data table.
`chr`	`character`(n): the chromosomes to query. When it's NULL, all CNEs will be returned.
`start`, `end`	`integer`(n): the start and end coordinates to fetch the CNEs.
`whichAssembly`	`character`(1): The coordinates to fetch CNEs are based on 'first' genome or 'last' genome.
`minLength`	`integer`(1): the minimal length for selected CNEs. The pair of CNEs must be both longer than this `minLength`.
`tAssemblyFn`, `qAssemblyFn`	`character`(1): filename of the 'twoBit' or 'fasta' file for the target and query genome.

Value

An object of `GRangePairs` is returned.

Author(s)

Ge Tan

Examples

```
dbName <- file.path(system.file("extdata", package="CNEr"),
                    "danRer10CNE.sqlite")
tableName <- "danRer10_hg38_45_50"

qAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
tAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")

## single chr, start, end
chr <- "chr6"
start <- 24000000L
end <-  27000000
minLength <- 50L
fetchedCNERanges <- readCNERangesFromSQLite(dbName, tableName, chr,
                                            start, end,
                                            whichAssembly="first",
                                            minLength=minLength,
                                            tAssemblyFn=tAssemblyFn,
                                            qAssemblyFn=qAssemblyFn)
## multiple chr, start, end
chr=c("chr1", "chr3")
start=c(90730248, 137523122)
end=c(90730300, 137523190)
fetchedCNERanges <- readCNERangesFromSQLite(dbName, tableName, chr,
                                            start, end,
                                            whichAssembly="second",
                                            minLength=minLength)

## chr, NULL, NULL
fetchedCNERanges <- readCNERangesFromSQLite(dbName, tableName, chr,
                                            start=NULL, end=NULL,
                                            whichAssembly="second",
                                            minLength=minLength)
```

`reverseCigar` *reverseCigar function*

Description

This function reverses the cigar string, i.e., 20M15I10D will be reversed to 10D15I20M.

Usage

```
reverseCigar(cigar, ops=CIGAR_OPS)
```

Arguments

`cigar`	A character vector of cigar strings.
`ops`	A character vector of the extended CIGAR operations. By default, CIGAR_OPS is used.

Value

A character vector contains the reversed cigar strings.

Author(s)

Ge Tan

See Also

cigar-utils

Examples

```
cigar = c("20M15I10D", "10D15I20M")
reverseCigar(cigar)
```

---

saveCNEToSQLite-methods

Save CNE to SQLite

Description

This function saves the CNE results into a local SQLite database.

Usage

```
saveCNEToSQLite(x, dbName, tableName=NULL, overwrite=FALSE)
```

Arguments

x	An object of CNE, with CNEFinal computed or a GRangePairs object.
dbName	character(1): the filename of the local SQLite database.
tableName	character(1): the name of table for this CNE data table. When it is NULL, the table name will be inferred from the assembly filenames and scanning window/identity, in the format of "danRer10_hg38_49_50".
overwrite	boolean(1): whether or not to overwrite the existing table.

Details

before loading into an SQLite database, a bin indexing system is used to index the CNE range, which provides faster SQL query.

Author(s)

Ge Tan

Examples

```
dbName <- tempfile()
data(cneFinalListDanRer10Hg38)
tableNames <- paste("danRer10", "hg38", names(cneFinalListDanRer10Hg38),
                    sep="_")
for(i in 1:length(cneFinalListDanRer10Hg38)){
  saveCNEToSQLite(cneFinalListDanRer10Hg38[[i]], dbName, tableNames[i],
                  overwrite=TRUE)
}
```

`scoringMatrix` *scoringMatrix*

Description

Generates the scoring matrix for *lastz* aligner.

Usage

```
scoringMatrix(distance = c("far", "medium", "near"))
```

Arguments

`distance`	It can be "far", "medium" or "close". It defines the scoring matrix used in *lastz* aligner. Generally, if two species are close to each other, for example human and chimp, "close" should be used. If two species have a divergence time of 100 MYA, "far" should be used. In other cases, "medium" should be used.

Value

A `matrix` of the scoring matrix is returned.

Note

HOXD70 is medium. HoxD55 is far. human-chimp.v2 is close.

Author(s)

Ge Tan

References

http://genomewiki.ucsc.edu/index.php/Hg38_17-way_conservation_lastz_parameters

See Also

lastz

Examples

```
scoringMatrix(distance="far")
```

subAxt-methods *Subset an* Axt *object*

Description

A 'subAxt' method for extracting a set of alignments from an Axt object.

Usage

```
subAxt(x, chr, start, end, select=c("target", "query"), qSize=NULL)
```

Arguments

x	An object of Axt.
chr	An object of character containing the names of the sequences in 'x' where to get the alignments from, or a GRanges object where 'start' and 'end' are missing. In the case of GRanges, the strand information is ignored.
start, end	An object of integer() or missing. These ranges should be based on the positive strand. When select is "query", the reverse complement alignments which lay inside this range will also be selected.
select	When select is 'target', the subset criteria are applied on target alignments in Axt. When select is 'query', the subset criteria are applied on query alignments in Axt.
qSize	integer(n): When select is 'query', 'qSize' must exist in 'x' or can be provided as a vector of chromosome lengths.

Details

Usually when we want to subset some axts from a Axt object, we care about all the axts within a certain range. The axts can come from the axt file with chr as reference (*i.e.*, target sequence), or the axt file with chr as query sequence. When the chr is query sequence, it can be on the negative strand. Hence, the size of chromosome is necessary to convert the search range to a range on negative strand coordinate.

When one Axt alignment partially overlaps the range, the whole Axt alignment will be extracted.

Value

An extracted Axt object is returned.

Author(s)

Ge Tan

See Also

psubAxt

Examples

```
library(GenomicRanges)
library(rtracklayer)

## Prepare the axt object
tAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Hsapiens.UCSC.hg38"),
                         "single_sequences.2bit")
qAssemblyFn <- file.path(system.file("extdata",
                         package="BSgenome.Drerio.UCSC.danRer10"),
                         "single_sequences.2bit")
axtFilesHg38DanRer10 <- file.path(system.file("extdata", package="CNEr"),
                                  "hg38.danRer10.net.axt")
axtHg38DanRer10 <- readAxt(axtFilesHg38DanRer10, tAssemblyFn, qAssemblyFn)

## "character", "integer", "integer" on "target" sequence
subAxt(axtHg38DanRer10, chr="chr1", start=148165963L, end=222131835L,
       select="target")

## "GRanges"" on "target" sequence
searchGRanges <- GRanges(seqnames="chr1",
                         ranges=IRanges(start=148165963L,
                                        end=222131835L),
                         strand="+")
subAxt(axtHg38DanRer10, searchGRanges, select="target")

## multiple "character", "integer", "integer" on "target" sequence
subAxt(axtHg38DanRer10, chr=c("chr1", "chr13"),
       start=c(148165963L, 94750629L),
       end=c(222131835L, 94966991L), select="target")

## "character" only on "target" sequence
subAxt(axtHg38DanRer10, chr="chr1", select="target")

## GRanges on "query" sequence
searchGRanges <- GRanges(seqnames="chr6",
                         ranges=IRanges(start=25825774,
                                        end=26745499),
                         strand="+")
subAxt(axtHg38DanRer10, searchGRanges, select="query")
```

---

summary *Utility functions related to Axt alignment*

Description

A collection of different functions used to deal with `Axt` object.

Usage

```
summary(object, ...) ## mismatch number and proportion
```

Arguments

object	An Axt object
...	Currently not used.

Value

A table object with the counts of mismatches, insertions, deletions and the matches of each base.

Author(s)

Ge Tan

Examples

```
axtFilesHg38DanRer10 <- file.path(system.file("extdata", package="CNEr"),
                                  "hg38.danRer10.net.axt")
axtHg38DanRer10 <- readAxt(axtFilesHg38DanRer10)
summary(axtHg38DanRer10)
```

---

syntenicDotplot-methods

Syntenic dotplot

Description

Syntenic dotplot for Axt alignment object or GRangePairs.

Usage

```
syntenicDotplot(x, firstSeqlengths=NULL, secondSeqlengths=NULL,
                firstChrs=NULL, secondChrs=NULL,
                col=c("blue", "red"), type=c("line", "dot"))
```

Arguments

x	Axt object: the whole genome pairwise alignment of two species under comparison or GRangePairs object.
firstSeqlengths, secondSeqlengths	integer(n): seqlengths for both the first (target) and second (query) genomes. When NULL, the seqlengths must exist in x.
firstChrs, secondChrs	character(n): the chromosomes to compare.
col	character(2): the colours for positive and negative strands.
type	"line" or "dot" plot type: When plotting massive number of ranges, "dot" should be used. Otherwise, "line" should be used.

Details

This syntenic dotplot is a type of scatter plot for `Axt` object, and line plot for `GRangePairs` object. In the case of possibly massive number of `Axt` alignments, the line plots will make it invisible at a large genome scale.

Each axis represents concatenated selected chromosomes laid end-to-end, and each dot in the scatter-plot represents a putative homologous match between the two genomes. These dotplots are used for whole genome comparisons within the same genome or across two genomes from different taxa in order to identify synteny.

Value

A `ggplot` object.

Note

For highly fragmented assemblies, the synteny is invisible on the dotplot.

Author(s)

Ge Tan

Examples

```
library(GenomeInfoDb)
library(BSgenome.Ggallus.UCSC.galGal3)
library(BSgenome.Hsapiens.UCSC.hg19)
## dotplot for Axt object
fn <- file.path(system.file("extdata", package="CNEr"),
                "chr4.hg19.galGal3.net.axt.gz")
axt <- readAxt(fn)
firstSeqlengths <- seqlengths(BSgenome.Hsapiens.UCSC.hg19)
secondSeqlengths <- seqlengths(BSgenome.Ggallus.UCSC.galGal3)
firstChrs <- c("chr4")
secondChrs <- c("chr4")
syntenicDotplot(axt, firstSeqlengths, secondSeqlengths,
                firstChrs=firstChrs, secondChrs=secondChrs,
                type="dot")

## dotplot for GRangePairs object
data(grangesPairsForDotplot)
syntenicDotplot(grangesPairsForDotplot, type="line")
```

---

writeAxt `writeAxt` *function*

Description

Write an axt object into a file.

Usage

```
writeAxt(axt, con)
```

Arguments

axt	An Axt object to write.
con	A connection object or a character string.

Author(s)

Ge Tan

See Also

readAxt

Examples

```
axtFile <- file.path(system.file("extdata", package="CNEr"),
                                 "hg38.danRer10.net.axt")
axt <- readAxt(axtFile)
writeAxt(axt, con=tempfile())
```

Index