Package: PostChicago (via r-universe)

November 25, 2026

Type Package

Title PostChicago - visualization and integration of Capture-(Hi)C data

Version 1.0.0

Description PostChicago is a toolbox for visualizing and assessing the output from the CHiCAGO pipeline. The plots created by PostChicago show reads or CHiCAGO scores over different regions. PostChicago can integrate different experiments with other types of datasets and compare separate conditions. Detailed description can be found in Krall et al. (2026) <doi:10.64898/2026.10.08.757510>.

biocViews Software, GeneRegulation, Epigenetics, Visualization

License Artistic-2.0

URL https://github.com/FeldmannLabDKFZ/PostChicago

Encoding UTF-8

RoxygenNote 7.3.3

Imports GenomicRanges, pheatmap, S4Vectors, IRanges, ggplot2, RColorBrewer, Chicago, utils, graphics, grDevices, stringr, gridExtra, BiocFileCache, matrixStats, rtracklayer, patchwork, dplyr, tidyr, rlang, data.table

Suggests knitr, rmarkdown, testthat (>= 3.0.0), BiocGenerics, codetools, BiocStyle, ragg

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Depends R (>= 4.4.0)

Config/pak/sysreqs cmake make libbz2-dev libicu-dev liblzma-dev libuv1-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:56:18 UTC

RemoteUrl https://github.com/bioc/PostChicago

RemoteRef RELEASE_3_24

RemoteSha 0c77ce53057a99dc59ce281c349d59cbb0ec52d0

Contents

aggregatePeaks . . . 2
aggregatePeaks_regions . . . 3
annotateInts . . . 5
baitmap2baited_genes . . . 6
boxplotsCapC . . . 6
getMatrix . . . 8
loadCdList . . . 10
loadGrl . . . 11
makeBedGraphs . . . 12
makeIntsTable . . . 14
makeOneGeneOnePeak . . . 16
makeQCplots . . . 17
plotAggregatePeaks . . . 18
plotInteractions . . . 20
plotStats . . . 23
quantifyWithinPeaks . . . 24
rmap2rmapgr . . . 26
runChicagoForPostChicago . . . 26

Index **29**

`aggregatePeaks` *Aggregate Interactions to Interaction Peaks*

Description

Takes a table of interactions (ints) and aggregates significant interactions within the distance dis in restriction fragments for each bait. Works similarly to the reduce() function from GenomicRanges. Caution: All interactions, regardless the sample from which they originated, are aggregated, creating a 'consensus' interaction peak set. If interaction peaks should be aggregated for individual samples, we recommend a prior sample-specific filtering of ints. Only creates the table if it is not yet saved, otherwise loads the existing table.

Usage

```
aggregatePeaks(ints, dis, samples, fileprefix = "ints", outdir = NULL)
```

Arguments

ints	Table containing all interactions that should be aggregated.
dis	Numeric, distance in restriction fragments over which interactions should be aggregated. dis=1 will merge two adjacent fragments with significant interactions.
samples	Character vector of sample names, should correspond to the sample names used in ints. Typically corresponding to the names provided in the list of ChicagoData objects cd from which ints has been generated.
fileprefix	Default: ints; Prefix of the filename for the aggregatePeaks table file, will be added the following extension: paste0('_aggregatePeaks_in_regions_',dis,'bp.txt').
outdir	Default: current directory; Directory where the final table is saved. Will not create a copy if already present.

Value

A table containing the positions of aggregated interactions. Redefines intIDs so that the ID contains all significant interactions that were aggregated. Data is automatically saved in a 'aggregatePeaks_dis' text file whose name is defined by fileprefix and dis. Also saved as a bedfile.

Examples

```
# example code:
extdata <- system.file("extdata", package="PostChicago")
outputDir <- file.path(extdata,'postchicago')

#read in the interactions table:
ints=read.delim(paste0(outputDir,'/ints_all.txt'),stringsAsFactors=FALSE)

#aggregate all peaks within a distance of 5 restriction fragments
dis <- 5
samples <- names(ints)[grep('_N$',names(ints))]
samples <- unlist(strsplit(samples,split ='_N'))
aggr <- aggregatePeaks(ints,dis,samples)
head(aggr)

dim(ints)
dim(aggr)
```

---

aggregatePeaks_regions *Aggregate Interactions to Interaction Peaks by Distance in bp*

Description

Takes a table of interactions (ints) and aggregates significant interactions within the distance dis in base pairs (bp) for each bait. Works similarly to the reduce() function from GenomicRanges. Caution: All interactions, regardless the sample from which they originated, are aggregated, creating a 'consensus' interaction peak set. If interaction peaks should be aggregated for individual samples, we recommend a prior sample-specific filtering of ints. Only creates the table if it is not yet saved, otherwise loads the existing table.

Usage

```
aggregatePeaks_regions(ints, dis, samples, fileprefix = "ints", outdir = NULL)
```

Arguments

ints	Table containing all interactions that should be aggregated.
dis	Numeric, distance in bp over which interactions should be aggregated. dis=1 will merge two adjacent fragments with significant interactions.
samples	Character vector of sample names, should correspond to the sample names used in ints. Typically corresponding to the names provided in the list of ChicagoData objects cd from which ints has been generated.
fileprefix	Default: ints; Prefix of the filename for the aggregatePeaks table file, will be added the following extension: paste0('_aggregatePeaks_in_regions_',dis,'bp.txt').
outdir	Default: current directory; Directory where the final table is saved. Will not create a copy if already present.

Value

A table containing the position of aggregated interactions. Redefines intIDs so that the ID of the first interacting fragment in a peak stands in as otherEnd. Data is automatically saved in a 'aggregatePeaks_dis' text file whose name is defined by fileprefix and dis. Also saved as a bedfile.

Examples

```
# example code:
extdata <- system.file("extdata", package="PostChicago")
outputDir <- file.path(extdata,'postchicago')

#read in the interactions table:
ints=read.delim(paste0(outputDir,'/ints_all.txt'),stringsAsFactors=FALSE)

#aggregate all peaks within a distance of 5 restriction fragments
dis <- 5000
samples <- names(ints)[grep('_N$',names(ints))]
samples <- unlist(strsplit(samples,split ='_N'))
aggr <- aggregatePeaks_regions(ints,dis,samples)
head(aggr)

dim(ints)
dim(aggr)
```

annotateInts *Annotate Interactions*

Description

Annotates interaction tables containing CaptureC data with features (typically ChIP-Seq peaks) from a GRangesList object (grl). Annotation is based on feature overlap.

Usage

```
annotateInts(ints, grl, minov = NULL, maxgp = NULL)
```

Arguments

ints	a table with interactions derived from ChicagoData objects by makeIntsTable(). Assumes six named columns: 'seqnames_bait','start_bait','end_bait' (coordinates of the baited restriction fragment) and 'seqnames_otherEnd','start_otherEnd','end_otherEnd' (coordinates of the interacting restriction fragment).
grl	GRangesList containing features with which ints should be annotated. Names of the grl correspond to the names that will be given to table columns.
minov	Default:NULL, minimum overlap (minoverlap in findOverlaps) between otherEnd or bait fragment and the feature.
maxgp	Default:NULL, maximum gap (maxgap in findOverlaps) between otherEnd or bait fragment and the feature.

Value

interactions file with annotations from GRanges file

Examples

```
# basic usage of annotateInts

##Create an interactions table
ints <- data.frame(seqnames_bait='chr1', start_bait=1, end_bait=1000,
                seqnames_otherEnd='chr1', start_otherEnd=10000, end_otherEnd=20000)
ints

##create a GRangesList with annotations
gr1 <- GenomicRanges::GRanges('chr1',IRanges::IRanges(10500,11000))
gr2 <- GenomicRanges::GRanges('chr1',IRanges::IRanges(900,1100))
grl <- GenomicRanges::GRangesList(gr1,gr2)
names(grl)=c('mytestpeaks1','mytestpeaks2')
grl

##Annotate interactions with intervals from grl
annotateInts(ints,grl)
```

baitmap2baited_genes *Convert baitmap to baited_genes*

Description

Searches for the Chicago baitmap file in 'designDir' and converts it to a GRanges object. Only executed if the output file 'baited_genes.rds' doesn't exist yet, otherwise loads the existing file.

Usage

```
baitmap2baited_genes(designDir = "designDir", save = FALSE)
```

Arguments

designDir	Path to the directory with Chicago design files. Default: 'designDir'
save	If TRUE, saves baited_genes as baited_genes.rds in designDir. Default: FALSE

Value

Returns a GenomicRanges object with the positions of the baits, annotated with fragment IDs (re_id) and gene names

Examples

```
# Example running baitmap2baited_genes
extdata <- system.file("extdata", package="PostChicago")
designDir <- file.path(extdata,'designDir')
baitmap2baited_genes(designDir,save=FALSE)
```

boxplotsCapC *Boxplots of summarized interactions data.*

Description

Boxplots of normalized interaction data quantified from either interaction peaks or oneGeneOnePeak tables. Returns 4 boxplots, containing downsampled read and Chicago score counts, total and size-normalized values. Function assumes that replicate data are saved in a column with the name structure '_rep1'.

Usage

```
boxplotsCapC(
  annotatedCapCTable,
  reps = FALSE,
  col = NULL,
  show_outliers = TRUE,
  show_notch = TRUE,
  name = NULL
)
```

Arguments

annotatedCapCTable	Table containing Capture-C quantification either over intervals or interaction peaks. Assumes the output from quantifyWithinPeaks().
reps	TRUE/FALSE(default); should replicate or summary data be plotted? Assumes replicate data are stored in columns names '_rep1', '_rep2' etc.
col	Custom plot color.
show_outliers	TRUE(default)/FALSE; Should outliers be shown?
show_notch	TRUE(default)/FALSE
name	optional, plot names

Value

Boxplots of scores and reads of each experiment of a specific feature

Examples

```
# example code

##Create an rmapgr object
rmapgr <- GenomicRanges::GRanges(rep(1,100), IRanges::IRanges( 1:100,2:101  ))
rmapgr$id=1:length(rmapgr)

##create interaction data table with two interactions
data <- data.frame(intID=c('1;2', '1;3,1;5'),
                   baitID = as.character(c(1,1)),
                   otherEndID=as.character(c(2, '3,5')))

##create a ChicagoData object with two samples
cdlist <- list(
  sample1 = data.frame(baitID=rep(1,11),
                       otherEndID=c(2:12),
                       N=c(sample(1:100,10,replace=TRUE),1000),
                       score=c(sample(0:10,10,replace=TRUE),100)),

  sample2 = data.frame(baitID=rep(1,11),
                       otherEndID=c(2:12),
                       N=sample(90:110,11,replace=TRUE),
```

```
                        score=sample(0:10,11,replace=TRUE))
)


##create replicate ChicagoData object from cdlist:
cdlist.reps <- c(cdlist,cdlist)
f<-function(x){
  x$N=x$N*sample((7:13)/10,11,replace=TRUE)
  x$score=x$score*sample((7:13)/10,11,replace=TRUE)
  return(x)
}
cdlist.reps=lapply(cdlist.reps,FUN=f)
names(cdlist.reps)=paste0(rep(names(cdlist),2),'_rep',rep(1:2,each=2))
cdlist.reps=cdlist.reps[order(names(cdlist.reps))]

##run quantifyWithinPeaks
annotatedCapCTable <- quantifyWithinPeaks(data, cdlist, cdlist.reps, rmapgr=rmapgr)

##plots
col=c('grey','red')
boxplotsCapC(annotatedCapCTable, col=col)
boxplotsCapC(annotatedCapCTable, col=col,show_notch=FALSE)
boxplotsCapC(annotatedCapCTable, col=rep(col,each=2),reps=TRUE)
```

`getMatrix` *Create Interaction Matrices*

Description

Function that creates and normalizes matrices containing reads or scores of fragment-to-fragment interactions surrounding the interactions that are provided in the interactions table ‘d‘.

Output is a list of matrices for the samples supplied in ‘L‘, including distance-matched control matrices. The matrix list is automatically saved in the ‘resfolder‘.

Matrices are stranded, such that the promoter is to the right (+ direction) of the interaction, which is important because the read numbers are increasing with increased promoter proximity. A new list is only created and saved as ‘.rds‘ file if the matrix cannot be found in the specified ‘resfolder‘, otherwise the preexisting matrix list is loaded. If the summit of an interaction peak or oneGeneOnePeak file is provided, getMatrix() extracts the summit info from the data, otherwise the function simply determines the central fragment of an interaction peak.

If norm = TRUE, normalized matrices are calculated from pre-exsisting unnormalized matrices. All values are normalized to the interaction in the reference sample defined by ‘normsam‘, which is recommended to be a positive control sample.

Usage

```
getMatrix(
  L,
```

```
  zoom = 100,
  d,
  resfolder = "matrices/",
  type = "reads",
  readnorm = "downsampling",
  name = "",
  norm = FALSE,
  normsam = 1,
  pseudo = 1,
  rmapgr = rmapgr
)
```

Arguments

L	list of ChicagoData tables.
zoom	Distance around interaction in restriction fragments for which matrices should be generated.
d	A table with cis-chromosomal interactions. Should contain the columns 'baitID', 'otherEndID', defined as in Chicago and can have a column 'intID', which is expected to be in the format 'baitID;otherEndID'. 'otherEndID' should be the center of interaction, from which the matrices will be calculated.
resfolder	Default: 'matrices/'. Path to the directory, where matrices should be stored. Will not create a copy if one is already saved in the given directory. Creates the directory if nonexistant.
type	'reads' (default)/'scores'. If reads, the function performs downsampling relative to the sample with the smallest amount of reads that are mapped to baits.
readnorm	Default: 'downsampling'. Read normalization, either 'downsampling' or 'scaling'. Scaling means the same type of normaliyation as for the lineplots in plotInteractions().
name	Name of the matrix.
norm	TRUE/FALSE (default). Should the data be normalized?
normsam	numeric; defines to which sample in the list L should the data be normalized. Default: 1 (the first sample!). Only used if norm=TRUE.
pseudo	Default: 1; pseudocount to be added during the normalization between matrices.
rmapgr	GenomicRanges object containing restriction fragments within the reference genome. IDs must be saved in a column called 'id'.

Value

List of interaction matrices.

Examples

```
# example code

##create ints table with 4 interactions
```

```
ints <- data.frame(intID=c('11;4', '11;7', '11;8','12;14'),
                   baitID = as.character(c(11,11,11,12)),
                   otherEndID=as.character(c(4,7,8,14)),
                   seqnames_bait=as.character((paste0('chr',rep(1,4)))),
                   start_bait=as.character(c(21,21,21,23)),
                   end_bait=as.character(c(22,22,22,23)),
                   seqnames_otherEnd=as.character((paste0('chr',rep(1,4)))),
                   start_otherEnd=as.character(c(7,13,15,27)),
                   end_otherEnd=as.character(c(8,14,16,28))
                   )

##make rmapgr
rmapgr <- GenomicRanges::GRanges(rep('chr1',20),
IRanges::IRanges( c(1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39),
c(2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40)  ) )
rmapgr$id=1:length(rmapgr)

cdlist <- list(
sample1 = data.frame(baitID=c(rep(11,20),rep(12,20)),
                     otherEndID=rep(c(1:20),2),
                     N=sample(1:20,40,replace=TRUE),
                     N.1=sample(1:20,40,replace=TRUE),
                     N.2=sample(1:20,40,replace=TRUE),
                     score=sample(1:5,40,replace=TRUE)),

sample2 = data.frame(baitID=c(rep(11,20),rep(12,20)),
                     otherEndID=rep(c(1:20),2),
                     N=sample(15:45,40,replace=TRUE),
                     N.1=sample(15:45,40,replace=TRUE),
                     N.2=sample(15:45,40,replace=TRUE),
                     score=sample(1:20,40,replace=TRUE))
      )

##getMatrix
getMatrix(L = cdlist, d = ints, zoom = 1, rmapgr=rmapgr)

##normalized matrices
getMatrix(L = cdlist, d = ints, zoom = 1, rmapgr=rmapgr,norm=TRUE,normsam=2)

##Add summit info to interactions:
ints$summit_re_id=c('4','7','8','13,14')
getMatrix(L = cdlist, d = ints, zoom = 1, rmapgr=rmapgr, resfolder='matrices_summits/')
```

`loadCdList` *Loads List with Interaction Tables*

Description

Reads in all ChicagoData objects in resultsDir and converts them to a list of ChicagoData tables. ChicagoData objects must be saved during the Chicago pipeline, which can be done by running Chicago via runChicagoForPostChicago()

Usage

```
loadCdList(resultsDir = "results", baited_genes = NULL)
```

Arguments

resultsDir	Default: 'results'; Directory containing ChicagoData output.
baited_genes	Optional; Used to control for the compatibility between loaded ChicagoData and current experiment; GenomicRanges object with at least two metadata columns: 1) 're_id' (captured fragment ID (bait), corresponding to fragment numbers indicated in rmapgr), 2) genenames' (names of the corresponding baits, typically gene names). Can be created automatically from the Chicago baitmap table (see R Bioconductor package 'Chicago') using the function baitmap2baited_genes().

Value

List containing ChicagoData tables.

Examples

```
# example code:
extdata <- system.file("extdata", package="PostChicago")
chicagoOutputDir <- file.path(extdata,'results')
designDir <- file.path(extdata,'designDir')

#create baited_genes:
baited_genes=baitmap2baited_genes(designDir,save=FALSE)

#load ChicagoData object list:
cdlist <- loadCdList(resultsDir = chicagoOutputDir, baited_genes = baited_genes)
```

---

loadGrl *Load Intervals into a GenomicRangesList*

Description

Reads in .rds and .bed files in intDir into a GRangesList object.

Usage

```
loadGrl(intDir = "intervals")
```

Arguments

intDir	Directory containing .rds or .bed files with intervals.

Value

GenomicRanges object grl containing all intervals

Examples

```
# example code

#requires precomputed matrices in the resfolder (see getMatrix())
extdata <- system.file("extdata", package="PostChicago")
intDir <- file.path(extdata,'intervals')
dir(intDir)

#Load intervals for which the boxplots should be quantified into a GRangesList
grl <- loadGrl(intDir = intDir)
```

---

makeBedGraphs *Create Bedgraphs From ChicagoData*

Description

Takes ChicagoData objects supplied in the list L, normalizes them to read coverage mapped to baits. Creates two file types: 1) a bedgraph file containing normalized reads for each sample mapping to the bait defined by id, 2) a bed file of the view point (bait).

Usage

```
makeBedGraphs(
  id,
  L_data,
  rmapgr,
  baited_genes,
  ints = NULL,
  sample_specific_ints = FALSE,
  L_norm = NULL,
  outputDir = "bedgraph",
  smoothingWindow = "off"
)
```

Arguments

id	baitID, must correspond to the baitIDs defined in baited_genes and the supplied rmapgr object
L_data	List of ChicagoData (cd) objects.
rmapgr	GenomicRanges object containing restriction fragments. IDs must be saved in a column called 'id'.
baited_genes	GenomicRanges object with at least two metadata columns: 1) 're_id' (captured fragment ID (bait), corresponding to fragment numbers indicated in rmapgr), 2) genenames' (names of the corresponding baits, typically gene names). Can be created automatically from the Chicago baitmap table (see R Bioconductor package 'Chicago') using the function baitmap2baited_genes().

ints — optional; table containing all significant interactions across L_data, usually created using, makeIntsTable(). If supplied, a separate bed file containing all significant interactions mapping to the id is created

sample_specific_ints — Default: FALSE. If TRUE and ints is supplied, creates a bedfile with significant interactions for each sample (Chicago score cutoff=5).

L_norm — Separate list of ChicagoData tables which should be used to normalize samples. Useful if bedgraphs should be created only for a subset of promoters or genomic locations, but the normalization performed based on the whole dataset. If not provided, L_data is used for normalization.

outputDir — Default: 'bedgraph'. Output directory for the .bedgraphs, automatically created if not currently present.

smoothingWindow — Default: 'off'. Over how many restriction fragments should the data be smoothed? Not recommended for UCSC visualization, but can improve visualization on IGV.

Value

Saves bedgraphs and annotation bedfiles of interactions with id as view point for each sample in L_data.

Examples

```
# example code:

extdata <- system.file("extdata", package="PostChicago")
chicagoOutputDir <- file.path(extdata,'results')
designDir <- file.path(extdata,'designDir')

#create baited_genes and rmapgr files:
baited_genes <- baitmap2baited_genes(designDir,save=FALSE)
rmapgr <- rmap2rmapgr(designDir, save = FALSE)

#load ChicagoData object list:
cdlist <- loadCdList(resultsDir = chicagoOutputDir, baited_genes = baited_genes)

## extract all view points (baits):
baits <- unique(unlist(lapply(cdlist,
                              FUN = function(x) unique(x$baitID))))

## create bedgraphs for selected view points and store them in outdir:
outdir <- "bedgraphs"
dir.create(outdir)
makeBedGraphs(baits[1], cdlist, rmapgr,
               baited_genes = baited_genes,
               outputDir = outdir
)

list.files(outdir)
```

`makeIntsTable` *Create Interactions Table*

Description

Integrates all significant interactions from multiple datasets, as supplied by the list of ChicagoData tables in L. The table is populated with scores, raw and downsampled reads from pooled and (optionally) replicate data.

Usage

```
makeIntsTable(
  L,
  baited_genes,
  repscores = FALSE,
  LL = NULL,
  ngroups = 1,
  scorecut = 5,
  readcut = 1,
  mode = "score",
  intervals = NULL,
  outfolder = NULL,
  overwrite = FALSE,
  saveFile = TRUE
)
```

Arguments

`L`	list of ChicagoData objects containing summarized interactions.
`baited_genes`	GenomicRanges object with at least two metadata columns: 1) 're_id' (captured fragment ID (bait), corresponding to fragment numbers indicated in rmapgr), 2) genenames' (names of the corresponding baits, typically gene names). Can be created automatically from the Chicago baitmap table (see R Bioconductor package 'Chicago') using the function baitmap2baited_genes().
`repscores`	TRUE/FALSE (default): Should replicate scores be added? if TRUE, LL must be supplied, Default: FALSE
`LL`	optional: List of ChicagoData tables containing individual replicates of summarized interactions. Names are expected to follow the naming convention '_rep1', '_rep2',..., '_repn'
`ngroups`	Default: 1; number of groups in which unique view points (baits) should be processed. Efficient computing is achieved for a maximum of ca. 100-200 entries per group.
`scorecut`	Default: 5 (as recommended by Chicago); Chicago score cutoff for significant interactions. Sometimes useful to lower to 3.
`readcut`	Default: 1; Reads cutoff for significant interactions.

mode	Options: 'score', 'read' or 'both'; Default: 'score'; Which parameter should be used for the extraction of significant interactions?
intervals	optional: A GenomicRanges object with intervals with which the otherEnds should overlap. Useful if mode='reads', to reduce the number of interactions in ints.
outfolder	Default: current directory; Path to the directory in which the output is saved.
overwrite	Default: FALSE. If TRUE runs the function even if an ints_all.txt is present in the outputdir, otherwise loads ints_all.txt.
saveFile	Default: TRUE; saves ints_all.txt in outfolder.

Value

Table of all significant interactions.

Examples

```
# example code

##Create an rmapgr object
rmapgr <- GenomicRanges::GRanges(rep(1,5), IRanges::IRanges( 1:5,2:6))
rmapgr$id=1:5

##create baited_genes with one gene
baited_genes <- GenomicRanges::GRanges(1,IRanges::IRanges(1,2))
baited_genes$re_id=1
baited_genes$genename='testgene'

##create a list of ChicagoData objects with two samples
cdlist <- list(
sample1 = data.frame(baitID=rep(1,4),
                     otherEndID=c(2,3:5),
                     N=c(2,5,10,8),
                     N.1=c(2,5,10,8),
                     N.2=c(3,4,9,7),
                     score=c(5, 1, 2, 4)),

sample2 = data.frame(baitID=rep(1,4),
                     otherEndID=c(2,3:5),
                     N=c(20,3,1,6),
                     N.1=c(20,3,1,6),
                     N.2=c(17,4,2,3),
                     score=c(3, 5, 3, 5))
     )

##run makeIntsTable
ints = makeIntsTable(cdlist,baited_genes,saveFile=FALSE)
head(ints)
```

makeOneGeneOnePeak *Make one-Gene-one-Peak Interaction Tables.*

Description

This function takes the interaction table ints (typically containing all significant interactions) and intersects them with intervals provided in the GRangesList grl.

Each entry in grl receives a unique ID and is assigned only one interaction with each bait with which at least one significant interaction could be detected.

Within the resulting OneGeneOnePeak table, start_otherEnd' and 'end_otherEnd' correspond to the start of the first and the end of the last restriction fragment that overlap a particular interval. 'otherEndID' corresponds to all overlapped restriction fragments from rmapgr separated by commas ('1,2,3'). 'summit_re_id' corresponds to the restriction ID of the central restriction fragment within rmapgr that overlaps an interval. If two intervals overlap exactly the same set of restriction fragments from rmapgr, this will be visible in the output table column 'interval_id_<my_interval>' as two comma-separated ids.

OneGeneOnePeak tables for each interval within grl are saved in the output directory.

Usage

```
makeOneGeneOnePeak(grl, rmapgr, ints, maxgap = -1, folder = "oneGeneOnePeak")
```

Arguments

grl	GRangesList containing features for which oneGeneOnePeak tables should be made. Names of the grl correspond to the names that will be given to table columns.
rmapgr	GenomicRanges object containing restriction fragments from the reference genome. IDs must be saved in a column called 'id'.
ints	Interactions table containing all significant interactions and typically derived by makeIntsTable().
maxgap	Default: -1, full overlap; Maximum gap between the otherEnd and an interval to count as overlap (see findOverlaps()).
folder	Output folder. Default: 'oneGeneOnePeak'

Value

A list with oneGeneOnePeak interactions. Saves a total of four files, for each of the overlapping intervals in grl: 1) An annotated OneGeneOnePeak interaction table as .txt. 2) An interaction table containing all information about interacting intervals, added to each row of the original interaction table ints, (not oneGeneOnePeak yet!): 'ints_<myintervaltype>_interacting.txt' 3) A reference table containing all interacting intervals, their ids (interval_id) and covered re_ids. In this table, intervals are summarized to one interval if they overlap exactly the same restriction fragments: '<myintervaltype>_interacting_rmapFrag_annotated.txt' 4) A reference table containing all intervals with interval_id and summit_re_id: '<myintervaltype>_withID.txt' oneGeneOnePeak tables can next be annotated with quantitative data using quantifyWithinPeaks().

Examples

```
# example code

##create ints table with 4 interactions
ints <- data.frame(intID=c('1;2', '1;3', '1;4','2;8'),
                   baitID = as.character(c(1,1,1,2)),
                   otherEndID=as.character(c(2,3,4,8)),
                   seqnames_bait=as.character((paste0('chr',rep(1,4)))),
                   start_bait=as.character(c(1,1,1,2)),
                   end_bait=as.character(c(2,2,2,3)),
                   seqnames_otherEnd=as.character((paste0('chr',rep(1,4)))),
                   start_otherEnd=as.character(c(3,5,7,15)),
                   end_otherEnd=as.character(c(4,6,8,16))
                   )

##make rmapgr
rmapgr <- GenomicRanges::GRanges(rep('chr1',8),
              IRanges::IRanges( c(1,3,5,7,9,11,13,15),
              c(2,4,6,8,10,12,14,16)  ) )
              rmapgr$id=1:length(rmapgr)

##make grl with one interval type and two intervals
grl <- GenomicRanges::GRangesList(testintervals=GenomicRanges::GRanges(rep('chr1',3),
                   IRanges::IRanges(c(2,5,6), c(4,7,8))),
                   testintervals2=GenomicRanges::GRanges(rep('chr1',2),
                   IRanges::IRanges(c(5,11), c(10,13))),
                  testintervals3 = GenomicRanges::GRanges('chr1',IRanges::IRanges(1,2)))

##run makeOneGeneOnePeak where intervals have to fully overlap
ogopList <- makeOneGeneOnePeak(grl = grl,rmapgr = rmapgr, ints = ints, maxgap=-1)
```

`makeQCplots` *Quality Control Plots*

Description

Creates and saves correlative heatmaps of reads and scores from interaction tables. If the table has more than 50000 rows, a random subset of 50000 is selected for the individual interaction heatmaps. All interactions are considered for correlation heatmaps.

Usage

```
makeQCplots(ints, folder = NULL, fontsize = 10, display_numbers = FALSE)
```

Arguments

`ints`	Interactions Table containing all significant interactions. Typically created using makeIntsTable().

folder Default: Current folder; Path to the directory in which plots are saved.

fontsize Default: 10.

display_numbers
Default: FALSE. If TRUE, displays correlations within the plots.

Value

A pdf file containing QC heatmaps: pheatmaps_QC.pdf.

Examples

```
# example code:
extdata <- system.file("extdata", package="PostChicago")
inDir <- file.path(extdata,'postchicago')

#read in interactions:
ints=read.delim(paste0(inDir,'/ints_all.txt'),stringsAsFactors=FALSE)

#assess interactions:
makeQCplots(ints)
```

---

plotAggregatePeaks *Plot Aggregate Interaction Profiles*

Description

Takes a list of matrices and plots aggregate interaction signal, similar for metaprofiles in ChIP-Seq.

Usage

```
plotAggregatePeaks(
  Lm,
  mainprefix = NULL,
  col = NULL,
  ylim = NULL,
  lty = NULL,
  xlim = NULL,
  k = 3,
  ylab = "normalised reads",
  plotwhat = "all"
)
```

Arguments

Lm	List of matrices containing interaction reads or scores.
mainprefix	Optional; text added to the plot title.
col	Custom colors for aggregate plots. Default: 4 colors.
ylim	Optional
lty	Optional; Line type for the aggregate plots
xlim	Optional; should be added in restriction fragments surrounding the interval/peak center. Example: c(-50,50). Default: All restriction fragments represented in the matrix.
k	Numeric; over how many fragments should the running mean be calculated? Default: 3
ylab	Optional; Default: 'normalised reads'
plotwhat	Defines what types of plots will be plotted. Options are 'all','mean','median' 'mean' and 'median' plot running averages over k fragments. Default: 'all': Four plots representing mean and median read counts both smoothed and unsmoothed over k fragments.

Value

Plot containing aggregate profiles of interactions.

Examples

```
# example code

##create list of matrices:

##matrix1
m1<-matrix(c(c(1:10,11,9:0)*sample((8:1.2)/10,21,replace=TRUE),
            c(1:10,11,9:0)*sample((8:1.2)/10,21,replace=TRUE),
            c(1:10,11,9:0)*sample((8:1.2)/10,21,replace=TRUE)),nrow=3)

##matrix2
m2<-matrix(c(c(21:30,39,29:20)*sample((8:1.2)/10,21,replace=TRUE),
            c(21:30,39,29:20)*sample((8:1.2)/10,21,replace=TRUE),
            c(21:30,39,29:20)*sample((8:1.2)/10,21,replace=TRUE)),nrow=3)
rownames(m1)=c('1;2','2;3','3;4')
rownames(m2)=rownames(m1)

##matrix list:
Lm=list(m1=m1,m2=m2,m1_con=m1*0.2,m2_con=m2*0.1)

##plot smoothed median enrichment profiles:
plotAggregatePeaks(Lm,plotwhat='median', ylab='reads',
    col=rep(c('red','blue'),2), lty=c(1,1,3,3))

##plot smoothed median profiles, increasing the smoothing window:
plotAggregatePeaks(Lm,plotwhat='median',k=5,ylab='reads',
```

```
    col=rep(c('red','blue'),2), lty=c(1,1,3,3))

##plot smoothed mean profiles:
plotAggregatePeaks(Lm,plotwhat='mean',k=3,ylab='reads',
col=rep(c('red','blue'),2), lty=c(1,1,3,3))

##plot median and mean, smoothed and unsmoothed profiles:
par(mfrow=c(2,2))
plotAggregatePeaks(Lm,plotwhat='all',k=3,ylab='reads',
   col=rep(c('red','blue'),2), lty=c(1,1,3,3))
```

---

plotInteractions *Plot Lineplots of Interactions*

Description

Plots lineplots of normalized Capture-C read counts around one view point (bait) from one or many samples. Optional annotation with genomic intervals of choice and significant interactions.

Usage

```
plotInteractions(
  L,
  id,
  k,
  zoom = 2e+05,
  rmapgr,
  ylim = NULL,
  show.legend = TRUE,
  d = NULL,
  preselected = FALSE,
  name = NULL,
  intervals = NULL,
  show.legend.intervals = TRUE,
  xlim = NULL,
  col = NULL,
  colintervals = NULL,
  colints = NULL,
  troubleshooting = FALSE,
  lwd = 2,
  lty = NULL,
  show.legend.outside = FALSE,
  overwrite.par.settings = FALSE,
  cex.intervals = 0.7,
  cex.legend = 1
)
```

Arguments

L	List with the ChicagoData tables
id	View point (baitID), corresponds to a fragment id in rmapgr.
k	Smoothing factor across k restriction fragments (running average). It is highly recommended to initially plot the data over a range of k to reveal granularity and broader relationships.
zoom	Default: 200000; Area around the bait in +/-bp to plot.
rmapgr	GenomicRanges object containing restriction fragments from the reference genome. IDs must be saved in a column called 'id'.
ylim	optional; Plotting limit of the y axis.
show.legend	TRUE(default)/FALSE; should the sample legend be shown inside the plot area?
d	Default: FALSE; interactions table to highlight significant interactions as transparent rectangles (shading). By default, if d is given, highlights any interaction. If different categories of interactions should be highlighted, these are specified in a column called 'clusters_refined'. Example usage would be 'lost', 'gained' and 'stable' interactions. Up to 8 categories can be defined which then receive a different color each.
preselected	, Default: FALSE; are interactions in d preselected for the correct baitID? Useful in cases when ids in the interactions table do not match the ids in L.
name	optional; Name of the view point, Default: name = baitID.
intervals	optional; Intervals to be plotted, provided as a named GenomicRangesList, useful for annotation of the plot with ChIP-Seq data.
show.legend.intervals	TRUE(default)/FALSE; should the legend for intervals be shown?
xlim	optional; Plotting limit of the x axis, as xlim=c(30000,200000), overrides zoom.
col	Custom colors of the lines. Default: 'Okabe-Ito' with a max of 8 different samples.
colintervals	Custom colors for interval annotation.
colints	Custom colors for significant interaction shading. Per default, interactions are plotted using the transparency parameter alpha of 0.3. However, this can be overridden by providing the alpha parameter together with the color: col <- rgb(0,0,0,0.2) will use black with an opacity of 0.2.
troubleshooting	if TRUE will return messages showing how far the process is, Default: FALSE
lwd	line width, Default 2pt
lty	line type, Default: 1
show.legend.outside	If TRUE, shows the legend outside of the plot. Note that the total plot area then becomes wider. Default: FALSE
overwrite.par.settings	If TRUE overwrites the default par settings for show.legend.outside par(mfrow=c(1,2)) by user-provided settings. Only used if show.legend.outside=TRUE. Default: FALSE
cex.intervals	Defines the size of the intervals legend. Default: 0.7
cex.legend	Defines the size of the samples legend. Default: 1

Value

Line plots of Capture-C reads surrounding one view point.

Examples

```
# example code (See package Vignette for more examples!)

#define directories
extdata <- system.file("extdata", package="PostChicago")
chicagoOutputDir <- file.path(extdata,'results')
designDir <- file.path(extdata,'designDir')

#create baited_genes and rmapgr:
baited_genes=baitmap2baited_genes(designDir,save=FALSE)
rmapgr=rmap2rmapgr(designDir,save=FALSE)

#load list with ChicagoData objects:
cdlist <- loadCdList(resultsDir = chicagoOutputDir, baited_genes = baited_genes)
cdlist=cdlist[-grep('rep',names(cdlist))]

#standard plot:
col=c('indianred1','seagreen')
name='Hoxb3'
id=baited_genes[baited_genes$genename==name]$re_id
k=15
ylim=c(0,200)
plotInteractions(cdlist,id,k,ylim=ylim,show.legend = TRUE,name=name, rmapgr=rmapgr,col=col)

#vary line type and width:
plotInteractions(cdlist,id,k,ylim=ylim,show.legend = TRUE,name=name, rmapgr=rmapgr,col=col,
    lwd=1,lty=c(1,2))

#change zoom around the bait:
plotInteractions(cdlist,id,k,ylim=ylim,show.legend = TRUE,name=name, rmapgr=rmapgr,col=col,
    zoom=500000)

#change zoom and the number of restriction fragments over which the running mean is calculated:
plotInteractions(cdlist,id,k=61,ylim=ylim,show.legend = TRUE,name=name, rmapgr=rmapgr,col=col,
    zoom=500000)

#display significant interactions in the plot

#read in interactions:
inDir <- file.path(extdata,'postchicago')
ints <- read.delim(paste0(inDir,'/ints_all.txt'),stringsAsFactors=FALSE)

#choose the color with which interactions should be highlighted
colints <- 'yellow'

#plot
plotInteractions(cdlist,id,k=31,ylim=ylim,show.legend = TRUE,name=name, rmapgr=rmapgr,col=col,
    zoom=200000, d = ints, colints=colints)
```

plotStats *Plot Interaction Statistics*

Description

Takes a list of ChicagoData tables and plots a per sample statistical summary.

The summary includes the total number of reads mapped to baits, total genes with significant interactions, total unique and significant interactions, the corresponding distributions per bait.

Usage

```
plotStats(
  L,
  col = NULL,
  lty = NULL,
  filename = NULL,
  plotDistribution = TRUE,
  plotExamples = FALSE,
  mode = "total"
)
```

Arguments

L	list of ChicagoData tables for which to extract statistics.
col	Custom colors for the plots. Default: Okabe-Ito, 8 colors.
lty	optional; vector defining line types.
filename	optional; if provided, the plots will be saved under filename; must have the extension '.pdf'
plotDistribution	if TRUE (default), distributions of reads and interactions per bait are plotted.
plotExamples	Default: FALSE; if TRUE, example distributions will be plotted based on the data from Mahara et al.
mode	one of 'total' or 'log'; Default: 'total'; if total, plots absolute read count distributions per significant interaction, if 'log' plots log2 read counts (pseudocount:1). Only relevant for own data, not for the example data.

Value

Plots with statistics of interactions for each sample and bait.

Examples

```
# example code:
extdata <- system.file("extdata", package="PostChicago")
dataPath <- file.path(extdata, 'dataPath')
chicagoOutputDir <- file.path(extdata,'results')
designDir <- file.path(extdata,'designDir')
intDir <- file.path(extdata,'intervals')
outputDir <- file.path(extdata,'postchicago')
baited_genes=baitmap2baited_genes(designDir,save=FALSE)
cdlist <- loadCdList(resultsDir = chicagoOutputDir, baited_genes = baited_genes)
L <- cdlist[-grep('rep',names(cdlist))]
plotStats(L,plotDistribution=FALSE,plotExamples=TRUE)  ##this plots example distributions
```

`quantifyWithinPeaks` *Quantification of Reads and Scores within oneGeneOnePeak or AggregatedPeak Interactions.*

Description

Quantifies reads and scores for interaction tables containing interactions overlapping multiple restriction fragments, such as oneGenOnePeak or aggregatePeak tables.

The function sums up all read and score information within the interaction, including restriction fragments that are not detected as significant interactions.

Output contains two types of quantifications:

1) total sum of reads (raw: _N and downsampled: _N_downsampled) and scores (_score) for each interaction

2) normalized quantifications (_N_downsampled_mean and _score_mean) that normalize for the total number of valid restriction fragments within an interaction peak.

Usage

```
quantifyWithinPeaks(
  data,
  cdlist,
  cdlist.reps = NULL,
  rmapgr,
  minsize = NULL,
  maxsize = NULL,
  file = NULL
)
```

Arguments

`data`	Table with interactions derived either through oneGeneOnePeak() or aggregatePeaks(). Usually contains the following columns: "intID","baitID","otherEndID","seqnames_bait","start_ minimal requirement: "intID","baitID","otherEndID".

cdlist	List of ChicagoData tables from which to make quantifications.
cdlist.reps	Optional: List of ChicagoData tables from which to make quantifications for individual replicates.
rmapgr	GenomicRanges object containing restriction fragments. IDs must be saved in a column called 'id'.
minsize	Optional, specifies the minimal size of valid restriction fragments and should correspond to the parameter chosen for creating the ChicagoData objects (mySettings$minFragLen) within the cdlist.
maxsize	Optional, specifies the maximal size of valid restriction fragments and should correspond to the parameter chosen for creating the ChicagoData objects (mySettings$maxFragLen) within the cdlist.
file	Optional; Path to the file in which the data should be saved. If specified, the final output will be saved under 'file', while up to four intermediate outputs will be saved under file1...4.txt. Note: Output will not be saved if file is not specified.

Value

Interaction table annotated with the sum and mean of read and score counts for each interaction.

Examples

```
# example code

##Create an rmapgr object
rmapgr <- GenomicRanges::GRanges(rep(1,5), IRanges::IRanges( 1:5,2:6  ))
rmapgr$id=1:5

##create data with two interactions
data <- data.frame(intID=c('1;2', '1;3,1;5'),
                   baitID = as.character(c(1,1)),
                   otherEndID=as.character(c(2, '3,5')))

##create a ChicagoData object with two samples
cdlist <- list(
sample1 = data.frame(baitID=rep(1,4),
                     otherEndID=c(2,3:5),
                     N=c(2,5,10,8),
                     score=c(5, 1, 2, 4)),

sample2 = data.frame(baitID=rep(1,4),
                     otherEndID=c(2,3:5),
                     N=c(20,3,1,6),
                     score=c(3, 5, 3, 5))
      )

##run quantifyWithinPeaks
quantifyWithinPeaks(data, cdlist, rmapgr=rmapgr)
```

rmap2rmapgr *Convert .rmap to the GenomicRanges object 'rmapgr'*

Description

Searches for the .rmap Chicago file in designDir and converts it to a GenomicRanges object rmapgr. Only executed if the GenomicRanges object doesn't exist in the design directory yet, otherwise loads the existing file.

Usage

```
rmap2rmapgr(designDir = "designDir", save = FALSE)
```

Arguments

designDir	Default: 'designDir'; Path to the directory with Chicago design files.
save	Default: FALSE; If TRUE, saves the converted object as .rds in the designDir, using the same name as the .rmap file.

Value

Returns a GenomicRanges object derived from .rmap.

Examples

```
# Example running rmap2rmapgr
extdata <- system.file("extdata", package="PostChicago")
designDir <- file.path(extdata,'designDir')
rmap2rmapgr(designDir,save=FALSE)
```

runChicagoForPostChicago
Run Chicago and save ChicagoData objects.

Description

Runs Chicago such that the saved output files are compatible with our pipeline. Default settings are only executed if the files are not yet saved in outputDir.

Usage

```
runChicagoForPostChicago(
  df,
  mySettings = NULL,
  outputDir = "results",
  DataPath = "dataPath",
  DesignDir = "designDir",
  overwrite = FALSE
)
```

Arguments

df	data.frame containing at least two columns: 'filename' (.chinput filename) and 'type' (sample type)
mySettings	Chicago Settings for the pipeline. If not specified, default Chicago settings will be used.
outputDir	Default 'results'; Output directory in which ChicagoData objects are saved.
DataPath	Default 'dataPath'; Directory of input files (typically .chinput).
DesignDir	Default 'designDir'; Directory of Experiment Design Files, see Chicago documentation.
overwrite	Default: FALSE; Overwrites ChicagoData objects that are present in outputDir if set to TRUE.

Value

Saves complete ChicagoData objects containing score and read counts for all ditags, including non-significant interactions and named after df$type. Also saves standard Chicago outputs in outputDir.

Examples

```
##Example code using package-supplied data
extdata <- system.file("extdata", package = "PostChicago")
chicagoOutputDir <- file.path(extdata,'results')
designDir <- file.path(extdata,'designDir')
dataPath <- file.path(extdata, "dataPath")
chicagoOutputDir <- file.path(extdata, "results")

## checking which .chinput files we have stored:
dir(dataPath)

##Adjust weights settings

weightsPath <- file.path(
system.file("extdata", package = "Chicago"),
"weights"
)
weightSettings <- file.path(weightsPath, "mESC-2reps.settings")
weightSettings <- read.delim(weightSettings, header = FALSE)
```

```
mySettings <- Chicago::defaultSettings()
mySettings[grep("weight", names(mySettings))] <- weightSettings[, 2]
mySettings$minFragLen <- 100

##Create a table with sample info

fls <- list.files(dataPath)
fls

df <- data.frame(
filename = fls,
RA = stringr::str_remove(fls, "_rep.*"),
rep = stringr::str_remove(stringr::str_remove(fls, "^.*h_"), "\\.chinput$")
)
df$type <- df$RA
head(df)

##Now you can run Chicago!

runChicagoForPostChicago(
 df = df, mySettings = mySettings, outputDir = chicagoOutputDir,
 DataPath = dataPath, DesignDir = designDir
)
```

Index

aggregatePeaks, 2
aggregatePeaks_regions, 3
annotateInts, 5

baitmap2baited_genes, 6
boxplotsCapC, 6

getMatrix, 8

loadCdList, 10
loadGrl, 11

makeBedGraphs, 12
makeIntsTable, 14
makeOneGeneOnePeak, 16
makeQCplots, 17

plotAggregatePeaks, 18
plotInteractions, 20
plotStats, 23

quantifyWithinPeaks, 24

rmap2rmapgr, 26
runChicagoForPostChicago, 26