Package 'recoup'

November 12, 2026

Type Package

Title An R package for the creation of complex genomic profile plots

Author Panagiotis Moulos <moulos@fleming.gr>

Maintainer Panagiotis Moulos <moulos@fleming.gr>

Depends R (>= 4.0.0), GenomicRanges, GenomicAlignments, ggplot2, ComplexHeatmap

Imports BiocGenerics, biomaRt, Biostrings, circlize, Seqinfo, GenomicFeatures, graphics, grDevices, httr, IRanges, methods, parallel, RSQLite, Rsamtools, rtracklayer, S4Vectors, stats, stringr, txdbmaker, utils

Suggests GenomeInfoDb, grid, BiocStyle, knitr, rmarkdown, zoo, RUnit, BiocManager, BSgenome, RMySQL

Description recoup calculates and plots signal profiles created from short sequence reads derived from Next Generation Sequencing technologies. The profiles provided are either sumarized curve profiles or heatmap profiles. Currently, recoup supports genomic profile plots for reads derived from ChIP-Seq and RNA-Seq experiments. The package uses ggplot2 and ComplexHeatmap graphics facilities for curve and heatmap coverage profiles respectively.

License GPL (>= 3)

Encoding UTF-8

LazyLoad yes

LazyData yes

URL https://github.com/pmoulos/recoup

biocViews ImmunoOncology, Software, GeneExpression, Preprocessing, QualityControl, RNASeq, ChIPSeq, Sequencing, Coverage, ATACSeq, ChipOnChip, Alignment, DataImport

VignetteBuilder knitr

Version 1.40.0

Date 2026-03-23

Collate 'annotation.R' 'argcheck.R' 'coverage.R' 'count.R' 'plot.R' 'profile.R' 'query.R' 'ranges.R' 'recoup.R' 'util.R'

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

git_branch RELEASE_3_23

git_last_commit 4a9a71f

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

buildAnnotationDatabase . . . 3
buildCustomAnnotation . . . 4
calcCoverage . . . 7
coverageRef . . . 8
coverageRnaRef . . . 9
getAnnotation . . . 10
getBiotypes . . . 11
getInstalledAnnotations . . . 12
importCustomAnnotation . . . 12
kmeansDesign . . . 14
loadAnnotation . . . 15
mergeRuns . . . 16
preprocessRanges . . . 17
profileMatrix . . . 19
recoup . . . 20
recoupCorrelation . . . 33
recoupHeatmap . . . 34
recoupPlot . . . 35
recoupProfile . . . 37
recoup_test_data . . . 38
removeData . . . 39
rpMatrix . . . 40
simpleGetSet . . . 41
sliceObj . . . 42

Index **45**

buildAnnotationDatabase

Build a local annotation database for recoup

Description

This function creates a local annotation database to be used with recoup so as to avoid long time on the fly annotation downloads and formatting.

Usage

```
buildAnnotationDatabase(organisms, sources,
db = file.path(system.file(package = "recoup"),
    "annotation.sqlite"),
forceDownload = TRUE, rc = NULL)
```

Arguments

organisms	a list of organisms and versions for which to download and build annotations. Check the main recoup help page for details on supported organisms and the Details section below.
sources	a character vector of public sources from which to download and build annotations. Check the main recoup help page for details on supported annotation sources.
db	a valid path (accessible at least by the current user) where the annotation database will be set up. It defaults to system.file(package = "recoup"), "annotation.sqlite") that is, the installation path of recoup package. See also Details.
forceDownload	by default, buildAnnotationDatabase will not download an existing annotation again (FALSE). Set to TRUE if you wish to update the annotation database for a particular version.
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization). Sometimes used for building certain annotation types.

Details

Regarding the organisms argument, it is a list with specific format which instructs buildAnnotationDatabase on which organisms and versions to download from the respective sources. Such a list may have the format: organisms=list(hg19=75, mm9=67, mm10=96:97) This is explained as follows:

- A database comprising the human genome versions hg19 and the mouse genome versions mm9, mm10 will be constructed.
- If "ensembl" is in sources, version 75 is downloaded for hg19 and versions 67, 96, 97 for mm9, mm10.
- If "ucsc" or "refseq" are in sources, the latest versions are downloaded and marked by the download date. As UCSC and RefSeq versions are not accessible in the same way as Ensembl, this procedure cannot always be replicated.

organisms can also be a character vector with organism names/versions (e.g. organisms = c("mm10","hg19")), then the latest versions are downloaded in the case of Ensembl.

Regarding db, this controls the location of the installation database. If the default is used, then there is no need to provide the local database path to any function that uses the database (e.g. the main recoup). Otherwise, the user will either have to provide this each time, or the annotation will have to be downloaded and used on-the-fly.

Value

The function does not return anything. Only the SQLite database is created or updated.

Author(s)

Panagiotis Moulos

Examples

```
# Build a test database with one genome
myDb <- file.path(tempdir(),"testann.sqlite")

organisms <- "mm10"
sources <- "ensembl"

# If the example is not running in a multicore system, rc is ignored
#buildAnnotationDatabase(organisms,sources,db=myDb,rc=0.5)

# A more complete case, don't run as example
# Since we are using Ensembl, we can also ask for a version
#organisms <- list(
#    mm9=67,
#    mm10=96:97,
#    hg19=75,
#    hg38=96:97
#)
#sources <- c("ensembl", "refseq")

## Build on the default location (depending on package location, it may
## require root/sudo)
#buildAnnotationDatabase(organisms,sources)

## Build on an alternative location
#myDb <- file.path(path.expand("~"),"my_ann.sqlite")
#buildAnnotationDatabase(organisms,sources,db=myDb)
```

---

buildCustomAnnotation *Import custom annotation to the recoup annotation database from GTF file*

Description

This function imports a GTF file with some custom annotation to the recoup annotation database.

Usage

```
buildCustomAnnotation(gtfFile, metadata,
db = file.path(system.file(package = "recoup"),
    "annotation.sqlite"), rewrite=TRUE)
```

Arguments

gtfFile	a GTF file containing the gene structure of the organism to be imported.
metadata	a list with additional information about the annotation to be imported. See Details.
db	a valid path (accessible at least by the current user) where the annotation database will be set up. It defaults to `system.file(package = "recoup"), "annotation.sqlite")` that is, the installation path of recoup package. See also Details.
rewrite	if custom annotation found, rwrite? (default FALSE). Set to TRUE if you wish to update the annotation database for a particular custom annotation.

Details

Regarding the `metadata` argument, it is a list with specific format which instructs `buildCustomAnnotation` on importing the custom annotation. Such a list may has the following members:

- `organism` a name of the organism which is imported (e.g. `"my_mm9"`). This is the only mandatory member.
- `source` a name of the source for this custom annotation (e.g. `"my_mouse_db"`). If not given or `NULL`, the word `"inhouse"` is used.
- `version` a string denoting the version. If not given or `NULL`, current date is used.
- `chromInfo` it can be one of the following:
  - a tab-delimited file with two columns, the first being the chromosome/sequence names and the second being the chromosome/sequence lengths.
  - a BAM file to read the header from and obtain the required information
  - a `data.frame` with one column with chromosome lengths and chromosome names as `rownames`.

See the examples below for a `metadata` example.

Regarding `db`, this controls the location of the installation database. If the default is used, then there is no need to provide the local database path to any function that uses the database (e.g. the main `metaseqr2`). Otherwise, the user will either have to provide this each time, or the annotation will have to be downloaded and used on-the-fly.

Value

The function does not return anything. Only the SQLite database is created or updated.

Author(s)

Panagiotis Moulos

Examples

```
# Dummy database as example
customDir <- file.path(tempdir(),"test_custom")
dir.create(customDir)

myDb <- file.path(customDir,"testann.sqlite")
chromInfo <- data.frame(length=c(1000L,2000L,1500L),
    row.names=c("A","B","C"))

# Build with the metadata list filled (you can also provide a version)
buildCustomAnnotation(
  gtfFile=file.path(system.file(package="recoup"),"dummy.gtf"),
  metadata=list(
    organism="dummy",
    source="dummy_db",
    version=1,
    chromInfo=chromInfo
  ),
  db=myDb
)

# Try to retrieve some data
myGenes <- loadAnnotation(genome="dummy",refdb="dummy_db",
  type="gene",db=myDb)
myGenes

## Real data!
## Setup a temporary directory to download files etc.
#customDir <- file.path(tempdir(),"test_custom")
#dir.create(customDir)

#myDb <- file.path(customDir,"testann.sqlite")

## Gene annotation dump from Ensembl
#download.file(paste0("ftp://ftp.ensembl.org/pub/release-98/gtf/",
#  "dasypus_novemcinctus/Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  file.path(customDir,"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"))

## Chromosome information will be provided from the following BAM file
## available from Ensembl
#bamForInfo <- paste0("ftp://ftp.ensembl.org/pub/release-98/bamcov/",
#  "dasypus_novemcinctus/genebuild/Dasnov3.broad.Ascending_Colon_5.1.bam")

## Build with the metadata list filled (you can also provide a version)
#buildCustomAnnotation(
#  gtfFile=file.path(customDir,"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  metadata=list(
#    organism="dasNov3_test",
```

```
#    source="ensembl_test",
#    chromInfo=bamForInfo
#  ),
#  db=myDb
#)

## Try to retrieve some data
#dasGenes <- loadAnnotation(genome="dasNov3_test",refdb="ensembl_test",
#  level="gene",type="gene",db=myDb)
#dasGenes
```

`calcCoverage` *Calculate coverages over a genomic region*

Description

This function returns a coverage list for the genomic regions in `mask` argument. Generally it should not be used alone and is intended for internal use, although it is useful for calculating stand-alone coverages.

Usage

```
calcCoverage(input, mask, strand = NULL,
    ignore.strand = TRUE, rc = NULL)
```

Arguments

`input`	a `GRanges` object or a list of `GRanges` (not a `GRangesList`!) or the path to a BAM or BigWig file.
`mask`	a `GRanges` or `GRangesList` object.
`strand`	see the `strandedParams` in the main `recoup` function.
`ignore.strand`	see the `strandedParams` in the main `recoup` function.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Details

`input` contains the short reads in one of the formats described in the arguments section. When `input` is a list, this list should contain one member per chromosome of the organism of interest.

`mask` contains the genomic regions over which the coverage will be calculated from the input reads. When calculating RNA-Seq profiles, `mask` must be a named `GRangesList` where each member represents the exons of the respective gene.

Value

A list of `Rle` objects representing the genomic coverages of interest.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverage Rle
mask <- makeGRangesFromDataFrame(df=test.genome,
    keep.extra.columns=TRUE)
small.cov <- calcCoverage(test.input[[1]]$ranges,mask)
```

`coverageRef` *Calculate coverage in a set of reference genomic regions (ChIP-Seq or unspliced mode)*

Description

This function fills the `coverage` field in the main `input` argument in `recoup` function.

Usage

```
coverageRef(input, mainRanges,
    strandedParams = list(strand=NULL, ignoreStrand=TRUE),
    rc = NULL)
```

Arguments

`input`	an input list as in `recoup` but with the `ranges` field of each member filled (e.g. after using `preprocessRanges`).
`mainRanges`	the genome from `recoup` as a `GRanges` object (e.g. the output from `makeGRangesFromDataFrame`).
`strandedParams`	see the `strandedParams` argument in the main `recoup` function.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Value

Same as input with the `ranges` fields filled.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
testGenomeRanges <- makeGRangesFromDataFrame(df=test.genome,
    keep.extra.columns=TRUE)
test.input <- coverageRef(
    test.input,
    mainRanges=testGenomeRanges
)
```

`coverageRnaRef` *Calculate coverage in a set of reference genomic regions (RNA-Seq or spliced mode)*

Description

This function is defunct! Please use `coverageRef`. This function fills the `coverage` field in the main `input` argument in `recoup` function.

Usage

```
coverageRnaRef( input, mainRanges,
    strandedParams = list(strand=NULL, ignoreStrand=TRUE),
    rc = NULL)
```

Arguments

`input`	an input list as in `recoup` but with the `ranges` field of each member filled (e.g. after using `preprocessRanges`).
`mainRanges`	a named `GRangesList` where list member names are genes and list members are `GRanges` representing each gene's exons.
`strandedParams`	see the `strandedParams` argument in the main `recoup` function.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Value

Same as input with the `ranges` fields filled.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
#data("recoup_test_data",package="recoup")

# Note: the figures that will be produced will not look
# realistic or pretty and will be "bumpy". This is because
# package size limitations posed by Bioconductor guidelines
# do not allow for a full test dataset. As a result, the input
# below is not an RNA-Seq dataset. Have a look at the
# vignette on how to test with more realistic data.

# Calculate coverages
#testGenomeRanges <- makeGRangesFromDataFrame(df=test.genome,
#    keep.extra.columns=TRUE)
#test.input <- coverageRef(
#    test.input,
#    mainRanges=test.exons
#)
```

`getAnnotation` *Annotation downloader*

Description

This function connects to the EBI's Biomart service using the package biomaRt and downloads annotation elements (gene co-ordinates, exon co-ordinates, gene identifications, biotypes etc.) for each of the supported organisms. See the help page of `recoup` for a list of supported organisms. The function downloads annotation for an organism genes or exons. It also uses the UCSC public database connection API to download UCSC and RefSeq annotations.

Usage

```
getAnnotation(org, type, refdb = "ensembl", ver = NULL,
    rc = NULL)
```

Arguments

`org`	the organism for which to download annotation. Check the main `recoup` help page for details on supported organisms.
`type`	either `"gene"` or `"exon"`.
`refdb`	the online source to use to fetch annotation. It can be `"ensembl"` (default), `"ucsc"` or `"refseq"`. In the later two cases, an SQL connection is opened with the UCSC public databases.
`ver`	the version of `org` to use as related to `refdb` or `NULL` for latest versions. See also the main `recoup` help page.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization). It is used in the case of `type="exon"` to process the return value of the query to the UCSC Genome Browser database.

Value

A data frame with the canonical (not isoforms!) genes or exons of the requested organism. When `type="genes"`, the data frame has the following columns: chromosome, start, end, gene_id, gc_content, strand, gene_name, biotype. When `type="exon"` the data frame has the following columns: chromosome, start, end, exon_id, gene_id, strand, gene_name, biotype. The gene_id and exon_id correspond to Ensembl gene and exon accessions respectively. The gene_name corresponds to HUGO nomenclature gene names.

Note

The data frame that is returned contains only "canonical" chromosomes for each organism. It does not contain haplotypes or random locations and does not contain chromosome M.

Author(s)

Panagiotis Moulos

Examples

```
#mm10.genes <- getAnnotation("mm10","gene")
```

`getBiotypes` *List default Ensembl biotypes*

Description

This function returns a character vector of Ensembl biotypes for each supported organism. Mostly for internal use, but can also be used to list the biotypes and use some of them to subset initial genomic regions to be profiled.

Usage

```
getBiotypes(org)
```

Arguments

`org`	One of the supported `recoup` organisms See `recoup` for further information.

Value

A character vector of biotypes.

Author(s)

Panagiotis Moulos

Examples

```
hg18.bt <- getBiotypes("hg18")
```

getInstalledAnnotations *Load a recoup annotation element*

Description

This function returns a data frame with information on locally installed, supported or custom, annotations.

Usage

```
getInstalledAnnotations(obj = NULL)
```

Arguments

obj	NULL or the path to a recoup SQLite annotation database. If NULL, the function will try to guess the location of the SQLite database.

Value

The function returns a data.frame object with the installed local annotations.

Author(s)

Panagiotis Moulos

Examples

```
db <- file.path(system.file(package="recoup"),
  "annotation.sqlite")
if (file.exists(db))
  ig <- getInstalledAnnotations(obj=db)
```

importCustomAnnotation *Import a recoup custom annotation element*

Description

This function imports GenomicRanges to be used with recoup from a local GTF file.

Usage

```
importCustomAnnotation(gtfFile, metadata,
    type = c("gene", "exon", "utr"))
```

Arguments

gtfFile	a GTF file containing the gene structure of the organism to be imported.
metadata	a list with additional information about the annotation to be imported. The same as in the buildCustomAnnotation man page.
type	one of the "gene", "exon" or "utr".

Value

The function returns a GenomicRanges object with the requested annotation.

Author(s)

Panagiotis Moulos

Examples

```
# Dummy GTF as example
chromInfo <- data.frame(length=c(1000L,2000L,1500L),
    row.names=c("A","B","C"))

# Build with the metadata list filled (you can also provide a version)
myGenes <- importCustomAnnotation(
  gtfFile=file.path(system.file(package="recoup"),"dummy.gtf"),
  metadata=list(
    organism="dummy",
    source="dummy_db",
    version=1,
    chromInfo=chromInfo
  ),
  type="gene"
)

## Real data!
## Gene annotation dump from Ensembl
#download.file(paste0("ftp://ftp.ensembl.org/pub/release-98/gtf/",
#  "dasypus_novemcinctus/Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  file.path(tempdir(),"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"))

## Build with the metadata list filled (you can also provide a version)
#dasGenes <- importCustomAnnotation(
#  gtfFile=file.path(tempdir(),"Dasypus_novemcinctus.Dasnov3.0.98.gtf.gz"),
#  metadata=list(
#    organism="dasNov3_test",
#    source="ensembl_test"
#  ),
#  type="gene"
#)
```

kmeansDesign *Apply k-means clustering to profile data*

Description

This function performs k-means clustering on recoup generated profile matrices and stores the result as a factor in the design element. If no design is present, then one is created from the k-means result.

Usage

```
kmeansDesign(input, design = NULL, kmParams)
```

Arguments

input	a list object created from recoup or partially processed by recoup or its data member. See the main input to recoup for further information.
design	See the respective argument in recoup for further information
kmParams	Contains parameters for k-means clustering on profiles. See the respective argument in recoup for further information.

Value

The design data frame, either created from scratch or augmented by k-means clustering.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(1000,1000),
    selector=NULL,
    plotParams=list(plot=FALSE,profile=TRUE,
        heatmap=TRUE,device="x11"),
    rc=0.1
)
```

```
# Re-design based on k-means
kmParams=list(k=2,nstart=20,algorithm="MacQueen",iterMax=20,
    reference=NULL)
design <- kmeansDesign(test.tss$data,kmParams=kmParams)
```

---

loadAnnotation *Load a recoup annotation element*

Description

This function creates loads an annotation element from the local annotation database to be used with recoup. If the annotation is not found and the organism is supported, the annotation is created on the fly but not imported in the local database. Use buildAnnotationDatabase for this purpose.

Usage

```
loadAnnotation(genome, refdb,
    type = c("gene", "exon", "utr"), version="auto",
    db = file.path(system.file(package = "recoup"),
        "annotation.sqlite"), summarized = FALSE,
        asdf = FALSE, rc = NULL)
```

Arguments

genome	a recoup supported organisms or a custom, imported by the user, name. See also the main recoup man page.
refdb	a recoup supported annotation source or a custom, imported by the user, name. See also the main recoup man page.
type	one of the "gene", "exon" or "utr".
version	same as the version in recoup.
db	same as the db in buildAnnotationDatabase.
summarized	if TRUE, retrieve summarized, non-overlaping elements where appropriate (e.g. exons).
asdf	return the result as a data.frame (default FALSE).
rc	same as the rc in buildAnnotationDatabase.

Value

The function returns a GenomicRanges object with the requested annotation.

Author(s)

Panagiotis Moulos

Examples

```
db <- file.path(system.file(package="recoup"),
  "annotation.sqlite")
if (file.exists(db))
  gr <- loadAnnotation(genome="hg19",refdb="ensembl",
    type="gene",db=db)
```

`mergeRuns` *Merge recoup outputs of same type*

Description

This function accepts two or more `recoup` output objects holding single samples to a merged object so that all samples can be used together. This is useful when many coverages must be calulated/plotted and memory issues do not allow effective parallelization.

Usage

```
mergeRuns(..., withDesign = c("auto", "drop"),
    dropPlots = TRUE)
```

Arguments

`...`	one or more `recoup` output objects.
`withDesign`	one of `"auto"` (default) or `"drop"`. Determines how to merge designs. See details for further information.
`dropPlots`	if profile and/or heatmap plots are attached to the input object(s), they will be recalculated if `dropPlots=="TRUE"` (default) or dropped otherwise

.

Details

The `withDesign` argument controls what should be done if any input has an attached design. The default behavioir (`"auto"`) will try to do its best to preserve compatible designs. If one or more inputs have the same design, it will be applied to the rest of the samples. If there is only one design, it will be applied to all samples (if you don't want this to happen, choose `"drop"`). If more than one sample has an attached design but these are incompatible (different numbers of rows/rownames, columns/columnnames), then all designs are dropped. Obviously, `withDesign="drop"` drops all attached designs and the output object is free of a design data frame.

Value

A `recoup` output object with as many samples as in ....

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

test.input.shift <- test.input
names(test.input.shift) <- paste(names(test.input.shift),"_1",sep="")
test.input.shift[[1]]$id <- paste0(test.input.shift[[1]]$id,"_1")
test.input.shift[[1]]$ranges <-
    shift(test.input.shift[[1]]$ranges,100)
test.input.shift[[2]]$id <- paste0(test.input.shift[[2]]$id,"_1")
test.input.shift[[2]]$ranges <-
    shift(test.input.shift[[2]]$ranges,100)

test.tss.1 <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    rc=0.1
)

test.tss.2 <- recoup(
    test.input.shift,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    rc=0.1
)

test.tss <- mergeRuns(test.tss.1,test.tss.2)
```

---

preprocessRanges *Read and preprocess BAM/BED files to GRanges*

Description

This function reads the BAM/BED files present in the input list object and fills the `ranges` field of the latter. At the same time it takes care of certain preprocessing steps like normalization.

Usage

```
preprocessRanges(input, preprocessParams, genome,
    bamRanges=NULL, bamParams = NULL, rc = NULL)
```

Arguments

input	an input list as in recoup but with the ranges field of each member filled (e.g. after using preprocessRanges).
preprocessParams	see the preprocessParams argument in the main recoup function.
genome	see the genome argument in the main recoup function.
bamRanges	a GRanges object to mask the BAM/BED files to save time and space. If NULL, the whole file is read.
bamParams	see the bamParams argument in the main recoup function.
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization).

Value

This function fills the ranges field in the main input argument in recoup function.

Author(s)

Panagiotis Moulos

Examples

```
# This example only demonstrates the usage of the
# preprocessRanges function. The input BAM files
# included with the package will not produce
# realistic plots as they contain only a very small
# subset of the original data presented in the
# vignettes (50k reads). Please see recoup vignettes
# for further demonstrations.
test.in <- list(
    WT_H4K20me1=list(
        id="WT_H4K20me1",
        name="WT H4K20me1",
        file=system.file("extdata",
            "WT_H4K20me1_50kr.bam",
            package="recoup"),
        format="bam",
        color="#EE0000"
    ),
    Set8KO_H4K20me1=list(
        id="Set8KO_H4K20me1",
        name="Set8KO H4K20me1",
        file=system.file("extdata",
            "Set8KO_H4K20me1_50kr.bam",
            package="recoup"),
        format="bam",
        color="#00BB00"
    )
)
```

```
pp=list(
    normalize="none",
    spliceAction="split",
    spliceRemoveQ=0.75
)

test.in <- preprocessRanges(test.in,pp)
```

`profileMatrix` *Calculate final profile matrices for plotting*

Description

This function fills the `profile` field in the main `input` argument in `recoup` function by calculating profile matrices from coverages which will be used for plotting.

Usage

```
profileMatrix(input, flank, binParams, rc = NULL,
    .feNoSplit = FALSE)
```

Arguments

`input`	an input list as in `recoup` but with the `ranges` the `coverage` fields of each member filled (e.g. after using `preprocessRanges` and `coverageRef`).
`flank`	see the `flank` argument in the main `recoup` function.
`binParams`	see the `binParams` argument in the main `recoup` function.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).
`.feNoSplit`	Temporary internal variable. Do not change unless you know what you are doing!

Value

Same as input with the `profile` fields filled.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")
# Do some work
testGenomeRanges <- makeGRangesFromDataFrame(df=test.genome,
    keep.extra.columns=TRUE)
w <- width(testGenomeRanges)
testGenomeRanges <- promoters(testGenomeRanges,upstream=2000,downstream=0)
testGenomeRanges <- resize(testGenomeRanges,width=w+4000)
test.input <- coverageRef(
    test.input,
    mainRanges=testGenomeRanges
)
test.input <- profileMatrix(
    test.input,
    flank=c(2000,2000),
    binParams=list(flankBinSize=50,regionBinSize=150,
        sumStat="mean",interpolation="auto"),
    rc=0.1
)
```

`recoup` *Create genomic signal profiles in predefined or custom areas using short sequence reads*

Description

This function calculates and plots signal profiles created from short sequence reads derived from Next Generation Sequencing technologies. The profiles provided are either sumarized curve profiles or heatmap profiles. Currently, `recoup` supports genomic profile plots for reads derived from ChIP-Seq and RNA-Seq experiments. The function uses ggplot2 and ComplexHeatmap graphics facilities for curve and heatmap coverage profiles respectively. The output list object can be reused as input to this function which will automatically recognize which profile elements needto be recalculated, saving time.

Usage

```
recoup(
    input,
    design = NULL,
    region = c("genebody", "tss", "tes", "utr3", "custom"),
    type = c("chipseq", "rnaseq"),
    signal = c("coverage", "rpm"),
    genome = c("hg18", "hg19", "hg38", "mm9" ,"mm10","mm39",
        "rn5", "rn6", "dm3", "dm6", "danrer7", "danrer10",
        "pantro4", "pantro5", "susscr3", "susscr11",
        "ecucab3", "tair10"),
    version = "auto",
```

```
refdb = c("ensembl", "ucsc", "refseq"),
flank = c(2000, 2000),
onFlankFail = c("drop","trim"),
fraction = 1,
orderBy = list(
    what = c("none", "suma", "sumn",
    "maxa","maxn", "avga", "avgn", "hcn"),
    order = c("descending", "ascending"),
    custom = NULL
),
binParams = list(
    flankBinSize = 0,
    regionBinSize = 0,
    sumStat = c("mean", "median"),
    interpolation = c("auto", "spline", "linear",
        "neighborhood"),
    binType = c("variable", "fixed"),
    forceHeatmapBinning = TRUE,
    forcedBinSize = c(50, 200),
    chunking = FALSE
),
selector = NULL,
preprocessParams = list(
    fragLen = NA,
    cleanLevel = c(0, 1, 2, 3),
    normalize = c("none", "linear",
        "downsample","sampleto"),
    sampleTo = 1e+6,
    spliceAction = c("split", "keep", "remove"),
    spliceRemoveQ = 0.75,
    bedGenome = NA
),
plotParams = list(
    plot = TRUE,
    profile = TRUE,
    heatmap = TRUE,
    correlation = TRUE,
    signalScale = c("natural", "log2"),
    heatmapScale = c("common", "each" ),
    heatmapFactor = 1,
    corrScale = c("normalized", "each"),
    sumStat = c("mean", "median"),
    smooth = TRUE,
    corrSmoothPar = ifelse(is.null(design), 0.1,
        0.5),
    singleFacet = c("none", "wrap", "grid"),
    multiFacet = c("wrap", "grid"),
    singleFacetDirection = c("horizontal", "vertical"),
```

```
    conf = TRUE,
    device = c("x11", "png", "jpg", "tiff", "bmp",
        "pdf", "ps"),
    outputDir = ".",
    outputBase = NULL
),
saveParams = list(
    ranges = TRUE,
    coverage = TRUE,
    profile = TRUE,
    profilePlot = TRUE,
    heatmapPlot = TRUE,
    correlationPlot = TRUE
),
kmParams = list(
    k = 0,
    nstart = 20,
    algorithm = c("Hartigan-Wong",
        "Lloyd", "Forgy", "MacQueen"),
    iterMax = 20,
    reference = NULL
),
strandedParams = list(
    strand = NULL,
    ignoreStrand = TRUE
),
ggplotParams = list(
    title = element_text(size = 12),
    axis.title.x = element_text(size = 10,
        face = "bold"),
    axis.title.y = element_text(size = 10,
        face = "bold"),
    axis.text.x = element_text(size = 9,
        face = "bold"),
    axis.text.y = element_text(size = 10,
        face = "bold"),
    strip.text.x = element_text(size = 10,
        face = "bold"),
    strip.text.y = element_text(size = 10,
        face = "bold"),
    legend.position = "bottom",
    panel.spacing = grid::unit(1, "lines")
),
complexHeatmapParams = list(
    main = list(
        cluster_rows = ifelse(length(grep(
            "hc", orderBy$what)) > 0, TRUE, FALSE),
        cluster_columns = FALSE,
```

```
                column_title_gp = grid::gpar(fontsize = 10,
                    font = 2),
                show_row_names = FALSE,
                show_column_names = FALSE,
                heatmap_legend_param = list(
                    color_bar = "continuous"
                )
            ),
            group=list(
                cluster_rows = ifelse(length(grep(
                    "hc", orderBy$what)) > 0, TRUE, FALSE),
                cluster_columns = FALSE,
                column_title_gp = grid::gpar(fontsize = 10,
                    font = 2),
                show_row_names = FALSE,
                show_column_names = FALSE,
                row_title_gp = grid::gpar(fontsize = 8,
                    font = 2),
                gap = unit(5, "mm"),
                heatmap_legend_param = list(
                    color_bar = "continuous"
                )
            )
        ),
        bamParams = NULL,
        onTheFly = FALSE,
        localDb = file.path(system.file(package = "recoup"),
            "annotation.sqlite"),
        rc = NULL
    )
```

Arguments

input	the main input to recoup can be either a list or a configuration file (with essentially the same contents as the list). In case of list input, it is a list of n lists, where n the number of samples. See Details for the inner list contents. Alternatively, input can be a text tab delimited file with a specific header (the same fields as each inner list when input is a list) and one row for each sample. Again, see Details section for the field specifications.
design	either a data frame with grouping factors as columns (e.g. two grouping factors can be strand, and Ensembl biotype) or a tab delimited text file with the same content (grouping factors in columns). If a data frame, the row.names attribute must correspond to the names (e.g. rownames) of the genome argument, or be a superset or subset of them. If a file, the first column must correspond to the names (e.g. rownames) of the genome argument or be a superset or subset of them.
region	one of "tss", "tes", "genebody", "custom".
type	one of "chipseq", "rnaseq".

`signal` plots signal based on coverage (`"coverage"` default) or reads per million `"rpm"` (experimental!).

`genome` when `region` is `"tss"`, `"tes"` or `"genebody"`, `genome` can be one of `"hg38"`, `"hg19"`, `"hg18"`, `"mm10"`, `"mm9"`, `"dm3"`, `"rn5"`, `"danrer7"`, `"pantro4"`, `"susscr3"` for human, mouse, fruitfly, rat, zebrafish and chimpanzee genomes respectively. When when `region` is one of the above or `"custom"`, `genome` can be a tab delimited BED-like text file or a data frame.

`version` the version of `genome` to use as related to `source`. Either `"auto"` (default) or a numeric value representing an Ensembl version or the creation date of UCSC or RefSeq annotations.

`refdb` one of `"ensembl"`, `"ucsc"` or `"refseq"`. It will be used to retrieve genomic reference regions when `genome` argument is one of the supported organisms.

`flank` a vector of length two with the number of base pairs to flank upstream and downstream the `region`. Minimum flank is 0bp and maximum is 50kb. It is always expressed in bp.

`onFlankFail` action to be taken when flanking causes the requested plot genomic coordinates to go beyond the lengths of reference sequences (e.g. chromosomes). It can be `"drop"` (default) or `"trim"`. Note that trimming will cause the flanking and main regions to be merged in genebody plots and therefore possibly reducing plot resolution.

`region`Minimum flank is 0bp and maximum is 50kb. It is always expressed in bp.

`fraction` a number from 0 to 1 (default) denoting the fraction of total data to be used. See Details for further information.

`orderBy` a named list whose members control the order of the genomic regions (related to the `genome` and `region` arguments as they appear in heatmap profiles. The list has the following fields:

- `what`: one of `"none"` (default), `"suma"`, `"sumn"`, `"maxa"`, `"maxn"`, `"avga"`, `"avgn"`, `"hcn"`, where `n` in `"sumn"`, `"maxn"`, `"hcn"` is the index of the profile which sould be used as reference. See Details for further information.
- `order`: either `"descending"` (default) for ordering coverages from highest to lowest, or `"ascending"` for the opposite.
- `custom`: a numeric vector of custom values (e.g. RNA abundance) that will be used to sort all the profiles. If provided, `what` will be ignored. Defaults to `NULL`.

`binParams` a named list whose members control the resolution of the coverage profiles. The list has the following fields:

- `flankBinSize`: the number of intervals (bins) into which the upstream and downstream regions are split and the per-base coverage is averaged across. If `0` (default), no binning is performed and the profiles are calculated based at the base-pair level (the highest possible resolution).
- `regionBinSize`: the number of intervals (bins) into which the main region is split and the per-base coverage is averaged across. If `0` (default), no binning is performed and the profiles are calculated based at the base-pair level (the highest possible resolution).

- sumStat: the statistic which is used to summarize the bin coverage. Can be "mean" (default) or "median".
- interpolation: the interpolation method to be used for coverage interpolation when the reference regions are of unequal lengths (e.g. gene bodies) and the regionBinSize is larger than some of the former. Can be "auto" (default), "spline", "linear" or "neighborhood". See Details for further explanations of each option.
- binType: the type of bins (variable or fixed) when signal="rpm". It defaults to "variable". See Details for further info.
- forceHeatmapBinning: if TRUE (default) and the profile resolution is very high (see flankBinSize and regionBinSize above), binning is applied prior to heatmap profile generation, otherwise the heatmaps will be oversized and will take a lot of time to render. Set to FALSE if both flankBinSize and regionBinSize are not zero so as to avoid unecessary profile recalculations.
- forcedBinSize: a vector with two integers representing the flankBinSize and regionBinSize to be used with forceHeatmapBinning above.
- chunking: TRUE (default) or FALSE. When TRUE, recoup will try to chunk the data for profile matrix calculation.

See Details for a few further notes in the usage of binParams.

selector — a named list whose members control some subsetting abilities regarding the input reference genomic regions (genome argument) or NULL (default) when the genome argument is/may be custom. When list, it has the following fields:

- id: a vector of ids of the same type as those present in the genome file or organism type/version.
- bioype: a vector of Ensembl biotypes that will be used to filter the genome when the latter is one of the supported organisms. Not used when genome is a custom file.
- exonType: currently not used.

preprocessParams — a named list whose members control certain preprocessing steps applied to the GRanges objects obtained while or after reading the input BAM/BED files with short reads or BigWig files with processed signals. The list has the following fields:

- fragLen: the expected DNA fragment length. Reads will be extended (or truncated) to this size. Not used for RNA-Seq type plots.
- cleanLevel: integer from 0 (default) to 3, controlling read filtering level prior to profile generation. See details for further information.
- normalize: one of "none" (default), "linear", "downsample", "sampleto". Controls how the coverages are normalized across samples. See Details for explanation of these options.
- sampleTo: a fixed library size for downsampling to be used with "sampleto" option above. If a sample has less reads than this fixed size, it is silently reported as is.
- spliceAction: one of "split" (default), "keep", "remove". Controls the action to be performed with spliced reads in the case of RNA-Seq samples. See Details for explanation of these options.

- spliceRemoveQ: the quantile of putative joint spliced read length to be used for read filtering when spliceAction is "remove". See Details for further explanations.
- bedGenome: one of the supported genomes, as when reading from bed files, chromosomal lenghts are not available and must be retrieved with another way.

plotParams a named list whose members control profile (curve and heatmap) plotting parameters. The list has the following fields:

- plot: if set to TRUE (default), the plots created with the calculated profiles with recoup are displayed. Set to FALSE to plot later using the output object.
- profile: if set to TRUE (default), the average coverage profile across the genomic regions of preference is calculated. Set to FALSE to suppress this.
- heatmap: if set to TRUE (default), the coverage heatmap profile across the genomic regions of preference is calculated. Set to FALSE to suppress this.
- correlation: if set to TRUE (default), the plots created with the calculated coverage correlations are displayed. Set to FALSE to plot later using the output object.
- signalScale: one of "natural" (default) or "log2" to control the signal scale of the final coverage plots. Hint: use log2 scale for RNA-Seq profiles as it produces much smoother plots.
- heatmapScale: one of "common" (default) or "each". When "common", a common heatmap color scale is calculated for all samples. When "each", each heatmap has its own color scale.
- heatmapFactor: a positive numeric value by which the upper color scale limit of the heatmap profile is multiplied. Defaults to 1. See Details for further information.
- corrScale: either "normalized" (default) or "each". Controls the scale display in coverage correlation plots. See Details for further information.
- sumStat: the statistic which is used to summarize coverage matrices. Can be "mean" (default) or "median".
- smooth: if TRUE (default), the final curve profiles are smoothed using splines. Set to FALSE for no smoothing. If the reference genomic regions are many, the differences are minimal.
- corrSmoothPar: a numeric value between 0 and 1 which controls the smoothing of correlation plots. Its default value is controlled by the presence of design. See Details for further information.
- corrScale: either "normalized" (default) or "each". See Details for further information.
- singleFacet: how should ggplot2 should facet the profiles with 1-factor design and only one sample whose profile will be plotted. Can be "none" (default), "wrap" or "grid". When "none", no gridding is applied and design factors are distinguished by colour. With more than one design factors, the multiFacet option below is used.
- multiFacet: how should ggplot2 should facet the profiles with 1-factor design and more than one samples whose profile will be plotted. Can be

"wrap" (default) or "grid". 2 or 3 (3rd would be colour) factor designs are faceted with "grid".

- singleFacetDirection: if single facetting is requested, how should the panels be arranged, horizontally (default, "horizontal") or vertically ("vertical").
- conf: plot also confidence intervals using geom_ribbon in profile or correlation plots.
- device: the R plotting device to redirect the plots to. Can be "x11" (default), "png", "jpg", "tiff", "bmp", "pdf", "ps".
- outputDir: the directory to place profiles when the plotting device is not "x11". Defaults to ".".
- outputBase: the naming template for output files when the plotting device is not "x11". The extensions "_profile" and "_heatmap" will be appended to distinguish each plot type. Leave NULL (default) for automatic filename generation.

saveParams — a named list which controls the information to be stored in the recoup output list object. The list has the following fields:

- ranges: set to TRUE (default) to store the GRanges object obtained from the BAM/BED files. Set to FALSE for not saving. Not applicable when input is of type BigWig.
- coverage: set to TRUE (default) to store the Rle list object obtained from the coverage calculations. Set to FALSE for not saving.
- profile: set to TRUE (default) to store the profile matrices exracted from coverage summarizations. Set to FALSE for not saving. It must be present when using the recoup output in the plotting functions recoupProfile and recoupHeatmap.
- profilePlot: set to TRUE (default) to store ggplot object containing the average coverage plot. Set to FALSE for not saving. Must be TRUE if you wish to use the recoup output later with recoupPlot.
- heatmapPlot: set to TRUE (default) to store ComplexHeatmap object containing the coverage heatmap plot. Set to FALSE for not saving. Must be TRUE if you wish to use the recoup output later with recoupPlot.
- correlationPlot: set to TRUE (default) to store ggplot object containing coverage correlation plot. Set to FALSE for not saving. Must be TRUE if you wish to use the recoup output later with recoupPlot.

See the Details section for some additional information.

kmParams — a named list which controls the execution of k-means clustering using standard R base function kmeans otherwise. The list has the following fields:

- k: the number of clusters for k-means clustering. When 0 (default), no k-means clustering is performed.
- nstart: See kmeans.
- algorithm: See kmeans.
- iterMax: See the iter.max parameter in kmeans.
- reference: which profile to use as reference for the determination of clusters and ordering. The rest of the heatmaps will be ordered according to the reference clustering. It can be either a sample id or NULL (default). If the

latter, all the profile matrices are merged into one big matrix and k-means clustering is performed on that matrix.

The result of k-means clustering will be appended to `design` as an additional field. If `design` is `NULL`, it will be created and passed to the plotting functions.

`strandedParams` a named list which controls how strand information will be treated (if present). The list has the following fields:

- `strand`: if set to `NULL` (default) then reads from both strands are used from the input BAM/BED files. If `"+"` or `"-"`, then only the respective strands are used. Not applicable for input of type BigWig.
- `ignoreStrand`: `TRUE` (default) or `FALSE`. Passed to the `ignore.strand` argument in the `findOverlaps` function used during coverage calculations.

`ggplotParams` a named list with theme parameters passed to the `ggplot` function of the `ggplot2` package. See the documentation of `ggplot2` for further details. Only the parameters mentioned in the function call are used.

`complexHeatmapParams`

a named list with groups of parameters passed to the `Heatmap` function of the `ComplexHeatmap` package. The list has the following fields:

- `main`: `Heatmap` parameters applied to each non-split (according to `design`) heatmap. See the `recoup` function call for supported parameters and `Heatmap` for further details.
- `main`: `Heatmap` parameters applied to each split (according to `design`) heatmap. See the `recoup` function call for supported parameters and `Heatmap` for further details.

`bamParams` BAM file read parameters passed to `BamFile`. See the related function. Currently this is not used.

`onTheFly` Read short reads directly from BAM files when input contains paths to BAM files. In this case the storage of short reads in the output list object as a GRanges object is not possible and the final object becomes less reusable but the memory footprint is lower. Defaults to `FALSE`.

`localDb` local path with the annotation database. See also `buildAnnotationDatabase`.

`rc` fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Details

When `input` is a list, is should contain as many sublists as the number of samples. Each sublist must have at least the following fields:

- `id`: a unique identifier for each sample which should not contain whitespaces and preferably no special characters.
- `file`: the full path to the BAM, BED of BigWig file. If the path to the BAM is a hyperlink, the BAM file must be indexed. BigWigs are already indexed.
- `format`: one of `"bam"`, `"bed"` or `"bigwig"`.

Additionally, each sublist may also contain the following fields:

- name: a sample name which will appear in plots.
- color: either an R color (see the colors function) or a hexadecimal color (e.g. "#FF0000").

When input is a text file, this should be strictly tab-delimited (no other delimiters like comma), should contain a header with the same names (case sensitive) as the sublist fields above (id, file, format are mandatory and name, color are optional).

When genome is not one of the supported organisms, it should be a text tab delimited file (only tabs supported) with a header line, or a data frame, where the basic BED field must be present, that means that at least "chromosome", "start", "end", a unique identifier column and "strand" must be present, \ preferably in this order. This file is read in a data frame and then passed to the makeGRangesFromDataFrame function from the GenomicRanges package which takes care of the rest. See also the makeGRangesFromDataFrame's documentation. When genome is one of the supported organisms, recoup takes care of the rest.

The version argument controls what annotation version is used (when using local annotation after having built a store with buildAnnotationDatabase or when downloading on the fly). When "auto", it will use the latest annotation version for the selected source. So, if source="ensembl", it will use the latest installed or available version for the specified organism based on information retrieved from the biomaRt package. For example, for organism="hg19", it will be 91 at the point where this manual is written. If source="refseq", recoup will either use the latest downloaded annotation according to a timestamp in the directory structure or download and use the latest tables from UCSC on the fly. If an annotation version does not exist, recoup will throw an error and exit.

When region is "tss", the curve and heatmap profiles are centered around the TSS of the (gene) regions provided with the genome argument, flanked accordind to the flank argument. The same applied for region="tes" where the plots are centered around the transcription end site. When region is "genebody", the profiles consist of two flanking parts (upstream of the TSS and downstream of the TES) and a middle part consisting of the gene body coverage profile. The latter is constructed by creating a fixed number of intervals (bins) along each gene and averaging the coverage of each interval. In some extreme cases (e.g. for small genes), the number of bins may be larger than the gene length. In these cases, a few zeros are distributed randomly across the number of bins to reach the predefined number of gene body intervals. When region is "custom" the behavior depends on the custom regions length. If it contains single-base intervals (e.g. ChIP-Seq peak centers), then the behavior is similar to the TSS behavior above. If it contains genomic intervals of equal or unequal size, the behavior is similar to the gene body case.

The fraction parameter controls the total fraction of both total reads and genomic regions to be used for profile creation. This means that the total reads for each sample are randomly downsampled to fraction*100% of the original reads and the same applied to the input genomic regions. This practice is followed by similar packages (like ngs.plot) and serves the purpose of a quick overview of how the actual profiles look before profiling the whole genome.

Regarding the orderBy parameters, for the options of the what parameter "sum" type of options order profiles according to i) the sum of coverages of all samples in each genomic region when orderBy$what="suma" or ii) the sum of coverages of sample n (e.g. 2) in each genomic region when orderBy$what="sumn" (e.g. orderBy$what="sum1"). The same apply for the "max" type of options but this time the ordering is performed according to the position of the highest coverage in each genomic profile. Ties in the position of highest coverage are broken randomly and sorting is performed with the default R sort. Similarly for "avg" type of options, the ordering is performed according to the average total coverage of a reference region. For the "hc" type of options, hierarchical clustering is performed on the selected (n) reference profile (e.g. orderBy$what="hc1")

and this ordering is applied to the rest of the sample profiles. When what="none", no ordering is performed and the input order is used (genome argument). If any design is present through the design argument or k-means clustering is also performed (through the kmParams argument), the orderBy directives are applied to each sub-profile created by design or k-means clustering.

Regarding the flankBinSize field of binParams, it is used only when region="genebody" or region="custom" and the custom regions are not single-base regions. This happens as when the genomic regions to be profiled are single-base regions (e.g. TSSs or ChIP-Seq peak centers), these regions are merged with the flanking areas and alltogether form the main genomic region. In these cases, only the regionBinSize field value is used. Note that when type="rnaseq" or region="genebody" or region="custom" with non single-base regions the values of flankBinSize and regionBinSize offer a fine control over how the flanking and the main regions are presented in the profiles. For example, when flankBinSize=100 and regionBinSize=100 with a gene body profile plot, the outcome will look kind of "unrealistic" as the e.g. 2kb flanking regions will look very similar to the usually larger gene bodies. On the other hand if flankBinSize=50 and regionBinSize=200, this setting will create more "realistic" gene body profiles as the flanking regions will be squished and the gene body area will look expanded. Within the same parameter group is also interpolation. When working with reference regions of different lengths (e.g. gene bodies), it happens very often that their lengths are a little to a lot smaller than the number of bins into which they should be split and averaged in order to be able to create the average curve and heatmap profiles. recoup allows for dynamic resolutions by permitting to the user to set the number of bins into which genomic areas will be binned or by allowing a per-base resolution where possible. The interpolation parameter controls what happens in such cases. When "spline", the R function spline is used, with the default method, to produce a spline interpolation of the same size as the regionBinSize option and is used as the coverage for that region. When "linear", the procedure is the same as above but using approx. When "neighborhood", a number of NA values are distributed randomly across the small area coverage vector, excluding the first and the last two positions, in order to reach regionBinSize. Then, each NA position is filled with the mean value of the two values before and the two values after the NA position, with na.rm=TRUE. This method should be avoided when >20% of the values of the extended vector are NA's as it may cause a crash. However, it should be the most accurate one in the opposite case (few NA's). When "auto" (the default), a hybrid of "spline" and "neighborhood" is applied. If the NA's constitute more than 20% of the extended vector, "spline" is used, otherwise "neighborhood". None of the above is applied to regions of equal length as there is no need for that. Furthermore, the parameter binType within the same parameter group controls the type of bins that a genomic interval should be split to in order to effectively calculate realistic signals when signal="rpm". When "variable", the number of bins that each genomic interval is split to is proportional to the square root of its size (the square root smooths the region length distribution, otherwise many regions e.g. in the set of human genes/transcripts will end up in unit-size bins even though they can support larger resolutions). The final signal is interpolated to a length of regionBinSize or flankBinSize to produce the final plots. When "fixed", the genomic intervals are "pushed" to have regionBinSize or flankBinSize bins, but if the areas are not large enought, they may end-up to many unit-size bins which will inflate and oversmooth the signal. It may give better results if the regions where the profile is to be created are all large enough.

Regarding the usage of selector$id field, this requires some careful usage, as if the ids present there and the ids of the genome areas do not match, there will be no genomic regions left to calculate coverage profiles on and the program will crash.

Regarding the usage of the preprocessParams argument, the normalize field controls how the GRanges representing the reads extracted from BAM/BED files or the signals extracted from Big-

Wig files will be normalized. When "none", no normalization is applied and external normalization is assumed. When "linear", all the library sizes are divided by the maximum one and a normalization factor is calculated for each sample. The coverage of this sample across the input genomic regions is then multiplied by this factor. When "downsample", all libraries are downsampled to the minimum library size among samples. When "sampleto", all libraries are downsampled to a fixed number of reads. The sampleTo field of preprocessParams tells recoup the fixed number of reads to downsample all libraries when preprocessParams$normalize="sampleTo". It defaults to 1 million reads (1e+6). The spliceAction field of preprocessParams is used to control the action to be taken in the presence of RNA-Seq spliced reads (implies type="rnaseq"). When "keep", no action is performed regarding the spliced reads (represented as very long reads spanning intronic regions in the GRanges object). When "remove", these reads are excluded from coverage calculations according to their length as follows: firstly the length distribution of all reads lengths (using the width function for GRanges) is calculated. Then the quantile defined by the field spliceRemoveQ of preprocessParams is calculated and reads above the length corresponding to this quantile are excluded. When "splice", then splice junction information inferred from CIGAR strings (if) present in the BAM files is used to splice the longer reads and calculate real coverages. This option is not available with BED files, however, BED files can contain pre-spliced reads using for example BEDTools for conversion. It should also be noted that in the case of BigWig files, only linear normalization is supported as there is no information on raw reads. The cleanLevel field controls what filtering will be applied to the raw reads read from BAM/BED files prior to producing the signal track. It can have four values: 0 for no read processing/filtering, (use reads as they are, no uniqueness and no removal of unlocalized regions and mitochondrial DNA reads, unless filtered by the user before using recoup), 1 for removing unlocalized regions (e.g. chrU, hap, random etc.), 2 for removing reads of level 1 plus mitochondrial reads (chrM) and 3 for removing reads of level 2 plus using unique reads only. The default is level 0 (no filtering).

Regarding the heatmapFactor option of plotParams, it controls the color scale of the heatmap as follows: the default value (1) causes the extremes of the heatmap colors to be linearly and equally distributed across the actual coverage profile values. If set smaller than 1, the the upper extreme of the coverage values (which by default maps to the upper color point) is multiplied by this factor and this new value is set as the upper color break (limit). This has the effect of decreasing the brightness of the heatmap as color is saturated before reaching the maximum coverage value. If set greater than 1, then the heatmap brightness is increased. Regarding the correlation option of plotParams, if TRUE then recoup calculates average coverage values for each reference region (row-wise in the profile matrices) instead of the average coverage in each base of the reference regions (column-wise in the profile matrices). This is particularly useful for checking whether total genome profiles for some biological factor/condition correlate with each other. This potential correlation is becoming even clearer when orderBy$what is not "none". Regarding the corrScale option of plotParams, it controls whether the average coverage curves over the set of reference genomic regions (one average coverage vale per genomic region, note the difference with the profiles where the coverage is calculated over the genomic locations themselves) should be normalized to a 0-1 scale or not. This is particularly useful when plotting data from different libraries (e.g. PolII and H3K27me1 occupancy over gene bodies) where other types of normalization (e.g. read downsampling cannot be applied). Regarding the corrSmoothPar option of plotParams, it controls the smoothing parameter for coverage correlation curves. If design is present, spline smoothing is applied (smooth.spline) with spar=0.5 else lowess smoothing is applied (lowess) with f=0.1. corrSmoothPar controls the spar and f respectively.

Regarding the usage of saveParams argument, this is useful for several purposes: one is for re-using recoup without re-reading BAM/BED/BigWig files. If the ranges are present in the input object to

recoup, they are not re-calculated. If not stored, the memory/storage usage is reduced but the object can be used only for simply replotting the profiles using recoupProfile and/or recoupHeatmap functions.

As a note regarding parallel calculations, the number of cores assigned to recoup depends both on the number of cores and the available RAM in your system. The most RAM expensive part of recoup is currently the construction of binned profile matrices. If you have a lot of cores (e.g. 16) but less than 128Gb of RAM for this number of cores, you should avoid using all cores, especially with large BAM files. Half of them would be more appropriate.

Finally, the output list of recoup can be provided as input again to recoup with some input parameters changed. recoup will then automatically recognize what has been changed and recalculate some, all or none of the genomic region profiles, depending on what input parameters have changed. For example, if any of the ordering options change (e.g. from no profile ordering to k-means clustering), then no recalculations are performed and the process is very fast. If region binning is changed (binParams$flankBinSize or binParams$regionBinSize), then only profile matrices are recalculated and coverages are maintained. If any of the preprocessParams changes, this causes all object including the short reads to be reimported and profiles recalculated from the beginning.

Value

a named list with five members:

- data: the input argument if it was a list or the resulting list from the unexported internal readConfig function, with the ranges, coverage and profile fields filled according to saveParams. This data member can be used again as an argument to recoup. The coverage and profile fields will be recalculated according to recoup parameters but the ranges will be resued if the input files are not changed.
- design: the design data frame which is used to facet the profiles.
- plots: the ggplot2 and/or Heatmap objects created by recoup.
- callopts: the majority of recoup call parameters. Their storage serves the reuse of a recoup list object so that only certain elements of plots are recalculated.

Author(s)

Panagiotis Moulos

Examples

```
# Load some sample data
data("recoup_test_data",package="recoup")

# Note: the figures that will be produced will not look
# realistic and will be "bumpy". This is because package
# size limitations posed by Bioconductor guidelines do not
# allow for a full test dataset. Have a look at the
# vignette on how to test with more realistic data.

# TSS high resolution profile with no design
test.tss <- recoup(
```

```
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    rc=0.1
)

# Genebody low resolution profile with 2-factor design,
# wide genebody and more narrow flanking
test.gb <- recoup(
    test.input,
    design=test.design,
    region="genebody",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    binParams=list(flankBinSize=50,regionBinSize=150),
    orderBy=list(what="hc1"),
    selector=NULL,
    rc=0.1
)
```

`recoupCorrelation` *Plot (faceted) average genomic coverage correlations*

Description

This function takes as input argument and output object from `recoup` and creates the average genomic curve correlations according to the options present in the input object. It can be used with saved recoup outputs so as to recreate the plots without re-reading BAM/BED files and re-calculating coverages.

Usage

```
recoupCorrelation(recoupObj, samples = NULL, rc = NULL)
```

Arguments

`recoupObj`	a list object created from `recoup`.
`samples`	which samples to plot. Either numeric (denoting the sample indices) or sample ids. Defaults to `NULL` for all samples.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Value

The function returns the `recoupObj` with the slot for the correlation plot filled. See also the `recoupPlot`, `getr` and `setr` function.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    plotParams=list(profile=FALSE,correlation=TRUE,
        heatmap=FALSE),
    rc=0.1
)

# Plot coverage correlations
recoupCorrelation(test.tss,rc=0.1)
```

`recoupHeatmap` *Plot genomic coverage heatmaps*

Description

This function takes as input argument and output object from `recoup` and creates heatmaps depicting genomic coverages using the `ComplexHeatmap` package and the options present in the input object. It can be used with saved recoup outputs so as to recreate the plots without re-reading BAM/BED files and re-calculating coverages.

Usage

```
recoupHeatmap(recoupObj, samples = NULL, rc = NULL)
```

Arguments

recoupObj	a list object created from recoup.
samples	which samples to plot. Either numeric (denoting the sample indices) or sample ids. Defaults to NULL for all samples.
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization).

Value

The function returns the recoupObj with the slot for the profile plot filled. See also the recoupPlot, getr and setr function.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    plotParams=list(profile=FALSE,heatmap=FALSE),
    rc=0.1
)

# Plot coverage profiles
recoupHeatmap(test.tss,rc=0.1)
```

---

recoupPlot *Plot list objects returned by recoup*

Description

This function takes as input argument an output object from recoup and plots the ggplot2 and ComplexHeatmap objects stored there.

Usage

```
recoupPlot(recoupObj, what = c("profile", "heatmap",
    "correlation"), device = c("x11", "png", "jpg", "tiff",
    "bmp", "pdf", "ps"), outputDir = ".",
    outputBase = paste(vapply(recoupObj,
    function(x) return(x$data$id), character(1)),
    sep = "_"), mainh = 1, ...)
```

Arguments

recoupObj	a list object created from recoup.
what	one or more of "profile", "heatmap" or "correlation". See the plotParams in the main recoup function. A minimum valid version is provided for default plotting.
device	a valid R graphics device. See the plotParams in the main recoup function.
outputDir	a valid directory when device is not "x11". See the plotParams in the main recoup function.
outputBase	a valid file name to be used as basis when device is not "x11". See the plotParams in the main recoup function. Defaults to a concatenation of sample ids.
mainh	the reference heatmap for ordering operations. Normally, calculated in recoup. See also the draw function in the ComplexHeatmap package. Deafults to the first heatmap.
...	further parameters passed either to ggsave or the base graphics devices of R.

Value

This function does not returns anything, just plots the recoup plots.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    plotParams=list(plot=FALSE,profile=TRUE,
        heatmap=TRUE,device="x11"),
```

```
    rc=0.1
)

# Plot coverage profiles
recoupPlot(test.tss)
```

---

recoupProfile *Plot (faceted) average genomic coverage profiles*

Description

This function takes as input argument and output object from recoup and creates the average genomic curve profiles according to the options present in the input object. It can be used with saved recoup outputs so as to recreate the plots without re-reading BAM/BED files and re-calculating coverages.

Usage

```
recoupProfile(recoupObj, samples = NULL, rc = NULL)
```

Arguments

recoupObj	a list object created from recoup.
samples	which samples to plot. Either numeric (denoting the sample indices) or sample ids. Defaults to NULL for all samples.
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization).

Value

The function returns the recoupObj with the slot for the profile plot filled. See also the recoupPlot, getr and setr function.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
```

```
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    plotParams=list(profile=FALSE,heatmap=FALSE),
    rc=0.1
)

# Plot coverage profiles
recoupProfile(test.tss,rc=0.1)
```

`recoup_test_data` *Reference and genomic sample regions for recoup testing*

Description

The testing data package containes a small gene set, a design data frame, some genomic regions and an input object for testing of recoup with ChIP-Seq and RNA-Seq data. Specifically:

- `test.input`: A small data set which contains 10000 reads from H4K20me1 ChIP-Seq data from WT adult mice and Set8 (Pr-Set7) KO mice. The tissue is liver.
- `test.genome`: A small gene set (100 genes) and their coordinates from mouse mm9 chromosome 12.
- test.design: A data frame containing the 100 above genes categorized according to expression and strand.
- test.exons: A `GRangesList` containing the exons of the 100 above genes for use with `recoup` RNA-Seq mode.

Format

`data.frame` and `list` objects whose format is accepted by recoup.

Author(s)

Panagiotis Moulos

Source

Personal communication with the Talianids lab at BSRC 'Alexander Fleming'. Unpublished data.

removeData *Remove data from recoup list object*

Description

This function clears members of the recoup output object that must be cleared in order to apply a new set of parameters without completely rerunning recoup.

Usage

```
removeData(input, type = c("ranges", "coverage",
    "profile", "reference"))
```

Arguments

input	a list object created from recoup or its data member.
type	one of "ranges", "coverage", "profile", "reference".

Details

This function clears members of the recoup output object which typically take some time to be calculated but it is necessary to clean them if the user wants to change input parameters that cause recalculations of these members. For example, if the user changes the binParams, the profile matrices ("profile" object member) have to be recalculated.

type controls what data will be removed. "ranges" removes the reads imported from BAM/BED files. This is useful when for example the normalization method is changed. "coverage" removes the calculated coverages over the reference genomic regions. This is required again when the normalization method changes. "profile" removes the profile matrices derived from coverages. This is required for example when the binParams main argument changes. Finally, "reference" removes the genomic loci over which the calculations are taking place. This is required when the genome, refdb or version main arguments change.

Value

A list which is normally the output of recoup without the members that have been removed from it.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Before removing
```

```
names(test.input)

# Remove a member
test.input <- removeData(test.input,"ranges")

# Removed
names(test.input)
```

---

rpMatrix *Reads per million profile matrices for plotting*

Description

This function fills the `profile` field in the main `input` argument in `recoup` function by calculating profile matrices using reads per million (rpm) or reads per kb per million reads (rpkm) over binned genomic areas of interest, instead of genomic coverage signals. The profile matrices are used for later plotting.

Usage

```
rpMatrix(input, mainRanges, flank, binParams,
    strandedParams = list(strand = NULL, ignoreStrand = TRUE),
    rc = NULL)
```

Arguments

`input`	an input list as in `recoup` but with the `ranges` field of each member filled (e.g. after using `preprocessRanges`.
`mainRanges`	the genome from `recoup` as a `GRanges` object (e.g. the output from `makeGRangesFromDataFrame`).
`flank`	see the `flank` argument in the main `recoup` function.
`binParams`	see the `binParams` argument in the main `recoup` function.
`strandedParams`	see the `strandedParams` argument in the main `recoup` function.
`rc`	fraction (0-1) of cores to use in a multicore system. It defaults to `NULL` (no parallelization).

Details

Regarding the calculation of `rpm` and `rpkm` values, the calculations slightly differ from the default defintions of these measurements in the sense that they are also corrected for the bin lengths so as to acquire human-friendly values for plotting.

Note that the genomic ranges (BAM/BED files) must be imported before using this function (as per the default `recoup` pipeline). We plan to support streaming directly from BAM files in the future.

Value

Same as input with the `profile` fields filled.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")
# Do some work
testGenomeRanges <- makeGRangesFromDataFrame(df=test.genome,
    keep.extra.columns=TRUE)
w <- width(testGenomeRanges)
testGenomeRanges <- promoters(testGenomeRanges,upstream=2000,downstream=0)
testGenomeRanges <- resize(testGenomeRanges,width=w+4000)
test.input <- rpMatrix(
    test.input,
    mainRanges=testGenomeRanges,
    flank=c(2000,2000),
    binParams=list(flankBinSize=50,regionBinSize=150,binType="fixed",
        sumStat="mean",interpolation="spline"),
    rc=0.1
)
```

---

simpleGetSet *Get and set some reusable objects from a recoup object*

Description

The `getr` and `setr` functions are used to get several reusable/changeable objects of `recoup` or replcace them (e.g. when the user wishes to change some `ggplot` or `ComplexHeatmap` parameters manually in a plot, or change the heatmap profile ordering mode).

Usage

```
getr(obj, key = c("design", "profile", "heatmap",
    "correlation", "orderBy", "kmParams", "plotParams"))
setr(obj, key, value = NULL)
```

Arguments

`obj`	a list object created from `recoup`.
`key`	one of `"design"`, `"profile"`, `"heatmap"`, `"correlation"`, `orderBy`, `kmParams`, `plotParams`. For `"profile"`, `"heatmap"`, the respective plots are retrieved or changed according to which function is called. For `setr` it can (and preferably) be a named list of arguments to be changed in the `recoup` list object. The list names are the same as above. For the rest, see the main `recoup` man page.
`value`	a valid `ggplot` or `HeatmapList` object created from `recoupProfile` or `recoupHeatmap` `recoupCorrelation` when changing plots. Values for all other types are also checked for validity.

Value

For `getr`, the object asked to be retrieved. For `setr`, the `obj` with the respective slots filled or replaced with `value`.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
    selector=NULL,
    plotParams=list(plot=FALSE,profile=TRUE,
        heatmap=TRUE,device="x11"),
    rc=0.1
)

# Plot coverage profiles

# Get the curve profile plot
pp <- getr(test.tss,"profile")

# Change some ggplot parameter
pp <- pp +
    theme(axis.title.x=element_text(size=14))

# Store the new plot
test.tss <- setr(test.tss,"profile",pp)
## or even better
# test.tss <- setr(test.tss,list(profile=pp))
```

`sliceObj` *Subset recoup output list objects*

Description

This function takes as input argument an output object from `recoup` and subsets it according to the inputs `i,j,k`. The attached plots may or may not be recalculated. Other input parameters stores in `obj$callopts` are not changed apart from any `selector` option which is dropped. Note that

when slicing vertically (by j), the $coverage member of the input data (if present) is **not** sliced, but remains as is. You can drop it using removeData as it is used to recalculate profile matrices only if bin sizes are changed in a recoup call.

Usage

```
sliceObj(obj, i = NULL, j = NULL, k = NULL,
    dropPlots = FALSE, rc = NULL)
```

Arguments

obj	a list object created from recoup.
i	vector of numeric or character indices, corresponding to the index or rownames or names of reference genomic regions. The design object member will also be subset. If there is a selector attached to the input object (see recoup arguments) it will be dropped.
j	vector of numeric indices corresponding to the profile matrix vertical index (or base pair position or bin of base pairs) so as to subset the profile. The function will do its best to "guess" new plotting x-axis labels.
k	vector of numeric or character indices corresponding to sample index or sample names. These will be returned.
dropPlots	if profile and/or heatmap plots are attached to the input object, they will be recalculated if dropPlots=="TRUE" (default) or dropped otherwise
.	
rc	fraction (0-1) of cores to use in a multicore system. It defaults to NULL (no parallelization).

Value

A recoup list object, susbet according to i, j.

Author(s)

Panagiotis Moulos

Examples

```
# Load some data
data("recoup_test_data",package="recoup")

# Calculate coverages
test.tss <- recoup(
    test.input,
    design=NULL,
    region="tss",
    type="chipseq",
    genome=test.genome,
    flank=c(2000,2000),
```

```
    selector=NULL,
    plotParams=list(plot=FALSE,profile=TRUE,
        heatmap=TRUE,device="x11"),
    rc=0.1
)

# Plot coverage profiles
o <- sliceObj(test.tss,i=1:10,k=1)
```

Index

∗ **datasets**
    recoup_test_data, 38

approx, *30*

buildAnnotationDatabase, 3, *15*, *28, 29*
buildCustomAnnotation, 4, *13*

calcCoverage, 7
colors, *29*
coverageRef, 8, *9*, *19*
coverageRnaRef, 9

data.frame, *5*, *15*
draw, *36*

findOverlaps, *28*

getAnnotation, 10
getBiotypes, 11
getInstalledAnnotations, 12
getr *(*simpleGetSet*)*, 41
ggplot, *28*
GRanges, *25*

Heatmap, *28*

importCustomAnnotation, 12

kmeans, *27*
kmeansDesign, 14

loadAnnotation, 15
lowess, *31*

makeGRangesFromDataFrame, *8*, *29*, *40*
mergeRuns, 16

preprocessRanges, *8, 9*, 17, *18, 19*, *40*
profileMatrix, 19

recoup, *3*, *7–11*, *14–16*, *18, 19*, 20, *33–37*, *39–43*
recoup_test_data, 38
recoupCorrelation, 33, *41*
recoupHeatmap, *27*, *32*, 34, *41*
recoupPlot, *27*, 35
recoupProfile, *27*, *32*, 37, *41*
removeData, 39, *43*
rpMatrix, 40

setr *(*simpleGetSet*)*, 41
simpleGetSet, 41
sliceObj, 42
smooth.spline, *31*
sort, *29*
spline, *30*

test.design *(*recoup_test_data*)*, 38
test.exons *(*recoup_test_data*)*, 38
test.genome *(*recoup_test_data*)*, 38
test.input *(*recoup_test_data*)*, 38