

Package: ChIPseeker (via r-universe)

October 7, 2026

Type Package

Title ChIPseeker for ChIP peak Annotation, Comparison, and Visualization

Version 1.49.3

Maintainer Guangchuang Yu <guangchuangyu@gmail.com>

Description This package implements functions to retrieve the nearest genes around the peak, annotate genomic region of the peak, statistical methods for estimate the significance of overlap among ChIP peak data sets, and incorporate GEO database for user to compare the own dataset with those deposited in database. The comparison can be used to infer cooperative regulation and thus can be used to generate hypotheses. Several visualization functions are implemented to summarize the coverage of the peak experiment, average profile and heatmap of peaks binding to TSS regions, genomic annotation, distance to TSS, and overlap of peaks or genes.

Depends R (>= 3.5.0)

Imports AnnotationDbi, aplot, BiocGenerics, boot, dplyr, enrichplot, IRanges, GenomeInfoDb, GenomicRanges, GenomicFeatures, ggplot2, gplots, graphics, grDevices, gtools, magrittr, methods, plotrix, parallel, RColorBrewer, rlang, rtracklayer, S4Vectors, scales, stats, tibble, TxDb.Hsapiens.UCSC.hg19.knownGene, utils, yulab.utils (>= 0.2.0)

Suggests clusterProfiler, ggimage, ggplotify, ggupset, ggVennDiagram, knitr, org.Hs.eg.db, prettydoc, ReactomePA, rmarkdown, testthat, TxDb.Hsapiens.UCSC.hg38.knownGene

Remotes GuangchuangYu/enrichplot

URL <https://yulab-smu.top/contribution-knowledge-mining/>

BugReports <https://github.com/YuLab-SMU/ChIPseeker/issues>

Encoding UTF-8

VignetteBuilder knitr

ByteCompile true

License Artistic-2.0

biocViews Annotation, ChIPSeq, Software, Visualization,
MultipleComparison

Config/roxygen2/version 8.1.0

Config/pak/sysreqs

libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libbz2-dev libicu-
dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository <https://bioc.r-universe.dev>

Date/Publication 2026-10-05 10:09:17 UTC

RemoteUrl <https://github.com/bioc/ChIPseeker>

RemoteRef HEAD

RemoteSha df67ad44b3d5772439fb74d6c66fb6751bc14e7a

Contents

ChIPseeker-package	3
.	5
.ChIPseekerEnv	6
annotatePeak	7
arrange.GRanges	10
as.data.frame.csAnno	12
as.GRanges	13
combine_csAnno	14
covplot	15
csAnno-class	17
downloadGEObedFiles	18
downloadGSMbedFiles	18
dropAnno	19
enrichAnnoOverlap	19
enrichPeakOverlap	21
filter.GRanges	23
getAnnoStat	25
getBioRegion	26
getGeneAnno	27
getGEOgenomeVersion	29
getGEOInfo	29
getGEOspecies	30
getNearestFeatureIndicesAndDistances	30
getPromoters	32
getSampleFiles	33
getTagMatrix	34
getTagMatrix.binning.internal	36
info	37
makeBioRegionFromGranges	38
mutate.GRanges	39

overlap	41
peak_Profile_Heatmap	42
peakHeatmap	44
peakHeatmap_multiple_Sets	46
plotAnnoBar	48
plotAnnoBar.data.frame	49
plotAnnoPie	50
plotAnnoPie.csAnno	51
plotAvgProf	52
plotAvgProf2	54
plotDistToTSS	55
plotDistToTSS.data.frame	56
plotPeakProf	58
plotPeakProf2	61
readPeakFile	63
rename.GRanges	65
seq2gene	66
shuffle	68
subset.csAnno	69
tagHeatmap	70
upsetplot.csAnno	71
vennpie	73
vennpie.csAnno	74
vennplot	76
vennplot.peakfile	78
Index	80

ChIPseeker-package	<i>ChIPseeker: ChIP peak Annotation, Comparison, and Visualization</i>
--------------------	--

Description

ChIPseeker is an R/Bioconductor package designed for comprehensive analysis of ChIP-seq (Chromatin Immunoprecipitation followed by sequencing) data. The package provides a complete toolkit for annotating ChIP peaks, comparing datasets, and visualizing results to facilitate biological interpretation.

Details

ChIPseeker offers four main categories of functionality:

1. Peak Annotation:

- **annotatePeak**: Comprehensive peak annotation including nearest genes, genomic features (promoter, exon, intron, UTR, etc.), distance to TSS, and optional gene annotation from databases
- **seq2gene**: Many-to-many mapping of genomic regions to genes for functional enrichment analysis, considering host genes, promoter regions, and flanking genes

- [readPeakFile](#): Read peak files in various formats (BED, narrowPeak, broadPeak, etc.)

2. Visualization:

- [plotAnnoBar](#), [plotAnnoPie](#): Visualize genomic annotation distribution
- [plotDistToTSS](#): Distribution of peaks relative to transcription start sites
- [covplot](#): Coverage plot showing peak distribution across chromosomes
- [plotAvgProf](#), [plotPeakProf](#): Average profiles and peak profiles around TSS or gene body
- [tagHeatmap](#), [peakHeatmap](#): Heatmaps of peak binding patterns
- [vennplot](#), [upsetplot](#): Visualize overlap among multiple peak sets

3. Peak Comparison and Overlap Analysis:

- [enrichAnnoOverlap](#): Test significance of gene overlap between ChIP experiments based on nearest gene annotation
- [enrichPeakOverlap](#): Test significance of genomic coordinate overlap using permutation testing
- [vennplot](#), [vennpie](#): Visualize overlaps among peak sets
- [shuffle](#): Shuffle peak positions for permutation testing

4. GEO Database Integration:

- [downloadGEObedFiles](#), [downloadGSMbedFiles](#): Download ChIP-seq data from GEO database
- [getGEOInfo](#), [getGEOSpecies](#), [getGEOgenomeVersion](#): Query GEO database metadata
- Access to over 17,000 ChIP-seq datasets from GEO for comparison

Additional Features:

- [getTagMatrix](#): Build tag matrices for profile plotting
- [getPromoters](#): Extract promoter regions
- [getBioRegion](#): Define custom genomic regions
- dplyr verb extensions for GRanges objects: [filter](#), [mutate](#), [rename](#), [arrange](#)
- Support for liftOver conversion between genome assemblies

Main Workflow

The typical ChIPseeker workflow:

1. Read peak files using [readPeakFile](#)
2. Annotate peaks using [annotatePeak](#) to get nearest genes, genomic features, and distances
3. Visualize results using [plotAnnoBar](#), [plotDistToTSS](#), etc.
4. (Optional) Compare with other datasets using [enrichPeakOverlap](#) or download from GEO
5. (Optional) Map regions to genes using [seq2gene](#) for functional enrichment analysis

Key Classes

- [csAnno-class](#): Container class for annotated peaks with metadata and statistics

Citation

If you use ChIPseeker in published research, please cite:

- Q Wang, M Li, T Wu, L Zhan, L Li, M Chen, W Xie, Z Xie, E Hu, S Xu, G Yu. Exploring epigenomic datasets by ChIPseeker. *Current Protocols*, 2022, 2(10): e585.
- G Yu, LG Wang, QY He. ChIPseeker: an R/Bioconductor package for ChIP peak annotation, comparison and visualization. *Bioinformatics*, 2015, 31(14):2382-2383.

Author(s)

Maintainer: Guangchuang Yu <guangchuangyu@gmail.com>

Contributors: Ming Li, Qianwen Wang, Yun Yan, Hervé Pagès, Michael Kluge, Thomas Schwarzl, Zhougeng Xu, Chun-Hui Gao

See Also

Useful links:

- <https://yulab-smu.top/contribution-knowledge-mining/>
- Report bugs at <https://github.com/YuLab-SMU/ChIPseeker/issues>
- Bioconductor page: <https://bioconductor.org/packages/ChIPseeker>

Description

capture name of variable

Usage

```
.(..., .env = parent.frame())
```

Arguments

...	expression
.env	environment

Value

expression

Examples

```
x <- 1
eval(.(x)[[1]])
```

<code>.ChIPseekerEnv</code>	<i>Initialize and manage ChIPseeker cache environment</i>
-----------------------------	---

Description

The function initializes and maintains a cache for TxDb objects and their associated genomic features. It checks if the cached TxDb matches the provided one, and updates the cache if they differ or if forced to update.

Usage

```
.ChIPseekerEnv(TxDb, item = "ChIPseekerEnv", force = FALSE)
```

Arguments

<code>TxDb</code>	TxDb or EnsDb annotation object to cache
<code>item</code>	character, name of the cache item. Default is "ChIPseekerEnv"
<code>force</code>	logical, whether to force update of the TxDb in cache, clearing all cached features. Default is FALSE

Details

This function manages the caching system for TxDb objects and related genomic features to improve performance by avoiding redundant computations.

The function performs the following:

1. Retrieves or creates a cache item
2. If no TxDb is cached, stores the provided TxDb
3. If a TxDb exists in cache, compares metadata to check if it matches
4. Updates cache if TxDb differs or if force=TRUE
5. Prints the genome version being used

This caching mechanism allows the package to reuse computed features (exons, introns, transcripts, etc.) across multiple function calls when using the same TxDb object.

Value

Invisibly returns NULL. Prints messages about genome version usage

annotatePeak	<i>Annotate peaks with genomic features</i>
--------------	---

Description

This function annotates ChIP-seq peaks with genomic features including nearest genes, genomic annotation (promoter, exon, intron, etc.), distance to TSS, and optional gene annotation from annotation databases. It can also identify flanking genes within a specified distance from peaks.

Usage

```
annotatePeak(
  peak,
  tssRegion = c(-3000, 3000),
  TxDb = NULL,
  level = "transcript",
  assignGenomicAnnotation = TRUE,
  genomicAnnotationPriority = c("Promoter", "5UTR", "3UTR", "Exon", "Intron",
    "Downstream", "Intergenic"),
  annoDb = NULL,
  addFlankGeneInfo = FALSE,
  flankDistance = 5000,
  sameStrand = FALSE,
  ignoreOverlap = FALSE,
  ignoreUpstream = FALSE,
  ignoreDownstream = FALSE,
  overlap = "TSS",
  verbose = TRUE,
  columns = c("ENTREZID", "ENSEMBL", "SYMBOL", "GENENAME")
)
```

Arguments

peak	peak file (BED, narrowPeak, broadPeak, etc.) or GRanges object containing genomic ranges of peaks to be annotated
tssRegion	numeric vector of length 2 specifying the range of TSS region for promoter annotation. Default is c(-3000, 3000), meaning 3kb upstream and 3kb downstream of TSS. Negative values indicate upstream, positive values indicate downstream
TxDb	TxDb or EnsDb annotation object, or a GRanges object containing user-defined genomic features. If a GRanges object is provided, some annotation features (genomic annotation, gene annotation, flanking genes) will be disabled
level	character, one of "transcript" or "gene". Determines whether to annotate peaks at the transcript level or gene level. Default is "transcript"
assignGenomicAnnotation	logical, whether to assign genomic feature categories (Promoter, Exon, UTR, Intron, etc.) to peaks. Default is TRUE

genomicAnnotationPriority	character vector specifying the priority order of genomic annotations when a peak overlaps multiple features. Must be a permutation of c("Promoter", "5UTR", "3UTR", "Exon", "Intron", "Downstream", "Intergenic"). Default prioritizes Promoter > 5UTR > 3UTR > Exon > Intron > Downstream > Intergenic
annoDb	character, name of annotation database package (e.g., "org.Hs.eg.db") to add gene identifiers (ENSEMBL, SYMBOL, GENENAME, ENTREZID). If NULL, only gene IDs from TxDb will be included. Default is NULL
addFlankGeneInfo	logical, whether to identify and include all flanking genes within the specified distance from each peak. If TRUE, adds flank_geneIds and flank_gene_distances columns. Default is FALSE
flankDistance	numeric, distance (in base pairs) to extend on both sides of each peak when searching for flanking genes. Only used when addFlankGeneInfo=TRUE. Default is 5000
sameStrand	logical, whether to only consider genes/transcripts on the same strand as the peak when finding nearest features. If FALSE, searches both strands. Default is FALSE
ignoreOverlap	logical, whether to ignore overlaps between peaks and TSS/features when finding the nearest gene. If FALSE, overlapping features will be prioritized. Default is FALSE
ignoreUpstream	logical, if TRUE, only annotate genes at the 3' end (downstream) of the peak. This restricts annotation to genes that come after the peak. Default is FALSE
ignoreDownstream	logical, if TRUE, only annotate genes at the 5' end (upstream) of the peak. This restricts annotation to genes that come before the peak. Default is FALSE
overlap	character, one of "TSS" or "all". Determines how overlaps are detected: "TSS" only considers overlaps with the TSS point (transcription start site), while "all" considers overlaps with any part of the gene/transcript body. When overlap="all", any gene overlapping the peak will be reported as the nearest gene, regardless of whether the overlap is at the TSS region. Default is "TSS"
verbose	logical, whether to print progress messages during annotation. Default is TRUE
columns	character vector, names of columns to retrieve from the annotation database specified by annoDb. Common values include "ENTREZID", "ENSEMBL", "SYMBOL", "GENENAME". Default is c("ENTREZID", "ENSEMBL", "SYMBOL", "GENENAME")

Details

The function performs comprehensive peak annotation through several steps:

1. **Feature extraction**: Extracts gene or transcript features from the provided TxDb or EnsDb annotation object, or uses user-defined GRanges.
2. **Nearest feature identification**: Finds the nearest gene/transcript to each peak using `getNearestFeatureIndicesAnd` considering strand specificity and overlap options. The `overlap` parameter controls whether to consider overlaps with the entire gene body ("all") or only TSS regions ("TSS").

3. ****Distance calculation****: Computes the distance from each peak to the transcription start site (TSS) of the nearest feature, with positive values indicating downstream and negative values indicating upstream.
4. ****Genomic annotation****: Assigns genomic feature categories (Promoter-TSS, 5' UTR, 3' UTR, Exon, Intron, Downstream, Intergenic) based on the peak's location relative to gene structure. The priority order can be customized via `genomicAnnotationPriority`.
5. ****Gene annotation****: Optionally adds gene identifiers (ENSEMBL, SYMBOL, GENENAME, ENTREZID) from annotation databases using the `annoDb` parameter.
6. ****Flanking gene information****: Optionally identifies all genes within a specified flanking distance from each peak, providing comprehensive information about nearby genes.

The function returns a `csAnno` object containing the annotated peaks with all metadata columns. When `TxDb` is provided as a `GRanges` object, some features (genomic annotation, gene annotation, flanking genes) are disabled, and the function works with user-defined genomic features.

Value

A `csAnno` object containing the annotated peaks. The main annotation data is stored in the `anno` slot (a `GRanges` object) with the following metadata columns:

- All original columns from the input peak file/`GRanges` object
- `annotation`: Genomic feature category of the peak. Possible values are Promoter-TSS, 5' UTR, 3' UTR, Exon, Intron, Downstream, and Intergenic. Only present if `assignGenomicAnnotation=TRUE`
- `geneChr`: Chromosome of the nearest gene/transcript
- `geneStart`: Start position of the nearest gene/transcript
- `geneEnd`: End position of the nearest gene/transcript
- `geneLength`: Length of the nearest gene/transcript
- `geneStrand`: Strand of the nearest gene/transcript ("+" or "-")
- `geneId`: Gene identifier (Entrez ID for `TxDb`, or `gene_id` for `EnsDb`)
- `transcriptId`: Transcript identifier (only when `level="transcript"`)
- `distanceToTSS`: Distance from peak to the transcription start site (TSS) of the nearest gene. Positive values indicate downstream, negative values indicate upstream. Distance is 0 for overlapping peaks
- `ENSEMBL`: Ensembl gene ID (only if `annoDb` is provided)
- `SYMBOL`: Gene symbol (only if `annoDb` is provided)
- `GENENAME`: Full gene name (only if `annoDb` is provided)
- `ENTREZID`: Entrez gene ID (only if `annoDb` is provided and requested in columns)
- `flank_geneIds`: Semicolon-separated list of all flanking gene IDs within the specified distance (only if `addFlankGeneInfo=TRUE`)
- `flank_gene_distances`: Semicolon-separated list of distances from the peak to each flanking gene. A distance of 0 means that the peak overlaps the feature range; at `level="transcript"` the feature is the whole transcript, so peaks inside a transcript body get 0 even when their distance to the TSS is not 0. For non-overlapping peaks the signed distance to the feature's TSS is reported (positive = downstream, negative = upstream)

- flank_txIds: Semicolon-separated list of all flanking transcript IDs (only if addFlankGeneInfo=TRUE and level="transcript")

The csAnno object also contains slots for:

- tssRegion: The TSS region used for annotation
- level: The annotation level ("transcript" or "gene")
- hasGenomicAnnotation: Whether genomic annotation was assigned
- detailGenomicAnnotation: Detailed annotation matrix (if available)
- annoStat: Annotation statistics
- peakNum: Total number of peaks

Author(s)

G Yu

See Also

[plotAnnoBar](#) [plotAnnoPie](#) [plotDistToTSS](#)

Examples

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfile <- system.file("extdata", "sample_peaks.txt", package="ChIPseeker")
peakAnno <- annotatePeak(peakfile, tssRegion=c(-3000, 3000), TxDb=txdb)
peakAnno

## End(Not run)
```

arrange.GRanges

Arrange (sort) GRanges objects using dplyr syntax

Description

The function sorts GRanges objects by converting them to a data.frame, applying the dplyr arrange operation, and converting back to GRanges. This allows sorting by any metadata column or standard GRanges columns (seqnames, start, end, width, strand).

Usage

```
## S3 method for class 'GRanges'
arrange(.data, ..., .by_group = FALSE)
```

Arguments

<code>.data</code>	GRanges object to arrange
<code>...</code>	column names or expressions to sort by. Multiple columns can be specified, with earlier columns taking precedence. Use <code>desc()</code> to sort in descending order. Can use any metadata column or standard GRanges columns (seqnames, start, end, width, strand)
<code>.by_group</code>	logical, if TRUE, will sort first by grouping variable (if the data is grouped). Currently not used but included for API consistency. Default is FALSE

Details

This function extends `dplyr::arrange()` to work with GRanges objects, allowing users to sort genomic ranges by any column using familiar dplyr syntax while preserving the GRanges structure.

The function performs the following steps:

1. Converts GRanges to data.frame using `as.data.frame()`
2. Applies `dplyr::arrange()` with the specified sorting columns
3. Converts back to GRanges using `makeGRangesFromDataFrame()` with `keep.extra.columns=TRUE`

All genomic range information and metadata columns are preserved. Sorting is stable (preserves original order for ties). Multiple sorting columns are applied in order (first column is primary sort key).

Value

GRanges object with ranges sorted by the specified columns. All metadata columns are preserved. The sort order is stable (ties preserve original order)

Examples

```
## Not run:
## Sort by chromosome and start position (genomic order)
peak.gr <- peak.gr %>% arrange(seqnames, start)

## Sort by distance (descending - closest to TSS first)
peak.gr <- peak.gr %>% arrange(desc(abs(distanceToTSS)))

## Sort by annotation, then by distance
peak.gr <- peak.gr %>% arrange(annotation, distanceToTSS)

## Sort by score (descending), then by chromosome
peak.gr <- peak.gr %>% arrange(desc(score), seqnames)

## Complex sorting with multiple criteria
peak.gr <- peak.gr %>%
  arrange(
    annotation,
    desc(score),
    abs(distanceToTSS)
```

```
)
## End(Not run)
```

as.data.frame.csAnno *Convert csAnno object to data.frame*

Description

The function converts the anno slot (GRanges object) to a data.frame, preserving all metadata columns. This is useful for exporting data, performing data.frame operations, or using with functions that require data.frame input.

Usage

```
## S3 method for class 'csAnno'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	csAnno object to convert
row.names	character vector or NULL, row names for the resulting data.frame. If NULL, default row names are used. Default is NULL
optional	logical, kept for compatibility with generic method but should be omitted. Default is FALSE
...	additional parameters (currently unused)

Details

This function converts a csAnno object to a data.frame by extracting and converting the GRanges object containing annotated peaks.

Value

A data.frame containing all peak information with columns:

- Standard GRanges columns: seqnames, start, end, width, strand
- All metadata columns: annotation, geneId, distanceToTSS, geneChr, geneStart, geneEnd, etc.

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

See Also

[as.GRanges](#) for converting to GRanges object

Examples

```
## Not run:
peak.df <- as.data.frame(peakAnno)
## Export to CSV
write.csv(peak.df, "annotated_peaks.csv")

## End(Not run)
```

as.GRanges

Convert csAnno object to GRanges

Description

The function returns the anno slot from the csAnno object, which contains all annotated peaks as a GRanges object with metadata columns (annotation, geneId, distanceToTSS, etc.).

Usage

```
as.GRanges(x)
```

Arguments

x csAnno object to convert

Details

This function extracts the GRanges object containing annotated peaks from a csAnno object.

Value

GRanges object containing all annotated peaks with their metadata columns. This is the same as accessing x@anno directly

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

Examples

```
## Not run:
peak.gr <- as.GRanges(peakAnno)
## Equivalent to: peak.gr <- peakAnno@anno

## End(Not run)
```

combine_csAnno	<i>Combine multiple csAnno objects</i>
----------------	--

Description

The function merges multiple csAnno objects that were created with the same annotation parameters (TSS region, level, genomic annotation status). It combines all peaks, merges annotation statistics, and creates a unified summary. This is useful for comparing or combining results from multiple ChIP-seq experiments.

Usage

```
combine_csAnno(x, ...)
```

Arguments

x	csAnno object (first object to combine)
...	additional csAnno objects to combine. At least one additional object must be provided

Details

This function combines two or more csAnno objects into a single object, merging their peaks and aggregating annotation statistics.

The function performs the following operations:

1. Validates that all objects are csAnno instances
2. Checks that all objects have the same tssRegion, level, and hasGenomicAnnotation values
3. Combines all GRanges objects from the anno slots
4. Merges detailGenomicAnnotation data frames by row binding
5. Aggregates annotation statistics by merging and summing frequencies across all objects
6. Sums the total peak counts

The resulting object contains all peaks from all input objects with combined annotation statistics showing the overall distribution across genomic features.

Value

A new csAnno object containing:

- All peaks from all input objects combined
- Merged annotation statistics with aggregated frequencies
- Combined detail genomic annotations
- Sum of peak counts from all objects

The tssRegion, level, and hasGenomicAnnotation are inherited from the first object (all must be identical)

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

References

<https://github.com/YuLab-SMU/ChIPseeker/issues/157>

See Also

[annotatePeak](#) for creating csAnno objects

Examples

```
## Not run:
## Combine two peak annotation results
combined <- combine_csAnno(peakAnno1, peakAnno2, peakAnno3)

## Visualize combined statistics
plotAnnoBar(combined)

## End(Not run)
```

covplot

Plot peak coverage across chromosomes

Description

The function creates a coverage plot showing where peaks are located across chromosomes. It calculates coverage by counting overlapping peaks at each genomic position, and displays the results as a bar/rectangular plot with one panel per chromosome.

Usage

```
covplot(
  peak,
  weightCol = NULL,
  xlab = "Chromosome Size (bp)",
  ylab = "",
  title = "ChIP Peaks over Chromosomes",
  chrs = NULL,
  xlim = NULL,
  lower = 1,
  fill_color = "black"
)
```

Arguments

peak	peak file (BED format) or GRanges object, or a list of peak files/GRanges objects for comparing multiple datasets
weightCol	character, name of a metadata column in the GRanges object to use as weights for coverage calculation. If NULL, all peaks have equal weight. Default is NULL
xlab	character, label for x-axis. Default is "Chromosome Size (bp)"
ylab	character, label for y-axis. Default is "" (empty)
title	character, plot title. Default is "ChIP Peaks over Chromosomes"
chrs	character vector, selected chromosomes to plot. If NULL, all chromosomes in the data are plotted. Default is NULL
xlim	numeric vector of length 2, genomic range to plot (start, end). If NULL, the entire chromosome is shown. Note: this applies to all chromosomes if specified. Default is NULL
lower	numeric, lower cutoff for coverage signal. Regions with coverage below this value are not displayed. Default is 1
fill_color	character vector or single value. For single peak: a color name or hex code. For multiple peaks (list): either a vector of colors (one per peak set) or a palette name (e.g., "Set1", "Dark2") from RColorBrewer. The order of colors matches the order of peak sets. Default is "black"

Details

This function visualizes the coverage of ChIP-seq peaks across chromosomes, showing the density and distribution of peaks along the genome.

The function performs the following steps:

1. Loads peaks from file or uses provided GRanges object
2. Calculates coverage using coverage() function, optionally weighted by a metadata column
3. Filters coverage by the lower threshold to remove low signals
4. Aggregates coverage into contiguous regions
5. Creates a ggplot2 visualization with:
 - One facet per chromosome (if multiple chromosomes)
 - Rectangular bars showing coverage regions
 - Color coding for multiple peak sets (if input is a list)

For multiple peak sets (list input), the function automatically assigns colors and creates a grouped plot showing all sets together.

Value

A ggplot2 object that can be further customized or printed. The plot shows peak coverage as rectangular bars, with one panel per chromosome when multiple chromosomes are present

Author(s)

G Yu

csAnno-class

Class "csAnno"

Description

The csAnno class is the main output object from `annotatePeak()`. It stores annotated peaks as a GRanges object along with metadata about the annotation process and summary statistics. This object can be used for visualization, further analysis, and conversion to other formats.

Details

This class represents the output of ChIPseeker peak annotation. It contains annotated peaks along with their genomic annotations, nearest gene information, and annotation statistics.

The class contains the following slots:

- anno: GRanges object containing all annotated peaks with metadata columns (annotation, geneId, distanceToTSS, etc.)
- tssRegion: numeric vector of length 2 specifying the TSS region used for promoter annotation (e.g., c(-3000, 3000))
- level: character, annotation level - either "transcript" or "gene"
- hasGenomicAnnotation: logical, whether genomic feature annotations (Promoter, Exon, UTR, etc.) were assigned
- detailGenomicAnnotation: data.frame with logical columns indicating which genomic features each peak overlaps
- annoStat: data.frame containing annotation statistics showing the percentage of peaks in each genomic feature category
- peakNum: numeric, total number of peaks that were annotated

Slots

anno GRanges object containing annotated peaks with all metadata

tssRegion numeric vector of length 2, TSS region for promoter annotation

level character, annotation level ("transcript" or "gene")

hasGenomicAnnotation logical, whether genomic annotations were assigned

detailGenomicAnnotation data.frame, detailed genomic annotation matrix

annoStat data.frame, annotation statistics summary

peakNum numeric, total number of annotated peaks

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

See Also

[annotatePeak](#)

downloadGEObedFiles	<i>downloadGEObedFiles</i>
---------------------	----------------------------

Description

download all BED files of a particular genome version

Usage

```
downloadGEObedFiles(genome, destDir = getwd())
```

Arguments

genome	genome version
destDir	destination folder

Author(s)

G Yu

downloadGSMbedFiles	<i>downloadGSMbedFiles</i>
---------------------	----------------------------

Description

download BED supplementary files of a list of GSM accession numbers

Usage

```
downloadGSMbedFiles(GSM, destDir = getwd())
```

Arguments

GSM	GSM accession numbers
destDir	destination folder

Author(s)

G Yu

dropAnno	<i>dropAnno</i>
----------	-----------------

Description

dropAnno

Usage

dropAnno(csAnno, distanceToTSS_cutoff = 10000)

Arguments

csAnno output of annotatePeak
distanceToTSS_cutoff distance to TSS cutoff

Details

drop annotation exceeding distanceToTSS_cutoff

Value

csAnno object

Author(s)

Guangchuang Yu

enrichAnnoOverlap	<i>Calculate overlap significance of ChIP experiments based on nearest gene annotation</i>
-------------------	--

Description

The function compares the overlap of genes associated with query and target ChIP-seq peaks. It uses a hypergeometric test to determine if the number of overlapping genes is significantly higher than expected by chance, suggesting biological similarity between the experiments.

Usage

```
enrichAnnoOverlap(
  queryPeak,
  targetPeak,
  TxDb = NULL,
  pAdjustMethod = "BH",
  chainFile = NULL,
  distanceToTSS_cutoff = NULL
)
```

Arguments

queryPeak	character, path to query peak file (BED format), or GRanges object containing query peaks
targetPeak	character vector of target peak file paths, a folder containing bed files, or a list of GRanges objects. Multiple target peaks can be provided for comparison
TxDb	TxDb or EnsDb annotation object (already loaded in R). If NULL, uses TxDb.Hsapiens.UCSC.hg19.knownGene as default. Must be an already-loaded R object, not a file path. To load from a file, use GenomicFeatures::makeTxDbFromGFF(), AnnotationDbi::loadDb(), or similar functions first
pAdjustMethod	character, method for p-value adjustment. Default is "BH" (Benjamini-Hochberg). See p.adjust for options
chainFile	character, path to chain file for liftOver conversion if target peaks are in a different genome assembly. Default is NULL
distanceToTSS_cutoff	numeric, distance cutoff in base pairs. Peaks with absolute distance to TSS greater than this value will be excluded from the analysis. If NULL, all peaks are included. Default is NULL

Details

This function tests whether two ChIP-seq experiments share significantly more genes than expected by chance, based on the genes nearest to their peaks.

The function performs the following steps:

1. Annotates both query and target peaks to find their nearest genes
2. Optionally filters peaks by distance to TSS using distanceToTSS_cutoff
3. Counts the number of overlapping genes between query and target
4. Performs a hypergeometric test to calculate p-values
5. Adjusts p-values for multiple testing

The hypergeometric test parameters:

- White balls (m): number of unique genes in query peaks
- Black balls (n): total genes minus query genes
- Drawn (k): number of unique genes in target peaks
- Overlap (q): number of overlapping genes

Value

A data.frame with one row per target peak comparison, containing:

- qSample: name of the query sample
- tSample: name of the target sample(s)
- qLen: number of unique genes in query peaks
- tLen: number of unique genes in target peaks
- N_OL: number of overlapping genes. Note that this counts *genes*, not peaks, so it can be larger than the number of input peaks: one peak is annotated to its nearest gene, but the same gene is reported once per peak and a peak region may overlap several target genes (see issue #84)
- pvalue: p-value from hypergeometric test
- p.adjust: adjusted p-value

Author(s)

G Yu

enrichPeakOverlap	<i>Calculate overlap significance of ChIP experiments based on genomic coordinates</i>
-------------------	--

Description

The function compares the genomic overlap between query and target ChIP-seq peaks. It uses a permutation test approach: randomly shuffling target peaks across the genome and comparing the observed overlap to the distribution of overlaps from shuffled data. This provides a p-value indicating whether the observed overlap is significantly higher than expected by chance.

Usage

```
enrichPeakOverlap(  
  queryPeak,  
  targetPeak,  
  TxDb = NULL,  
  pAdjustMethod = "BH",  
  nShuffle = 1000,  
  chainFile = NULL,  
  pool = TRUE,  
  mc.cores = detectCores() - 1,  
  verbose = TRUE,  
  symmetric = FALSE  
)
```

Arguments

queryPeak	character, path to query peak file (BED format), or GRanges object containing query peaks
targetPeak	character vector of target peak file paths, a folder containing bed files, or a list of GRanges objects. Multiple target peaks can be provided
TxDb	TxDb or EnsDb annotation object (already loaded in R). Required for shuffling peaks (to get chromosome lengths). If NULL, uses TxDb.Hsapiens.UCSC.hg19.knownGene as default. Must be an already-loaded R object, not a file path
pAdjustMethod	character, method for p-value adjustment. Default is "BH" (Benjamini-Hochberg). See p.adjust for options
nShuffle	integer, number of permutations for the statistical test. More shuffles provide more accurate p-values but take longer. Default is 1000. Set to 0 to skip permutation testing (p-value will be NA)
chainFile	character, path to chain file for liftOver conversion if target peaks are in a different genome assembly. Default is NULL
pool	logical, whether to pool all target peaks together for a single test. If TRUE, tests overlap with the combined set of all target peaks. If FALSE, tests overlap with each target peak set separately. Default is TRUE
mc.cores	integer, number of CPU cores to use for parallel processing. Default is detectCores()-1. See mclapply
verbose	logical, whether to print progress messages. Default is TRUE
symmetric	logical, whether the p-value should be direction free. Default is FALSE, which reports the one-sided test described above. If TRUE, the mirrored direction (query and target exchanged) is computed as well and the two are combined as $\min(1, 2 \cdot \min(p, p_{\text{rev}}))$ (Hedges), so that the result does not depend on the order of the arguments. Note that this doubles the number of permutations and that the smallest reportable p-value becomes $2/(n\text{Shuffle}+1)$; use a larger nShuffle and set.seed() for a reproducible result (issue #84)

Details

This function tests whether two ChIP-seq experiments have significantly more overlapping peaks than expected by chance, using permutation testing with shuffled peak positions.

The function performs the following steps:

1. Loads query and target peaks (supports BED files or GRanges objects)
2. Optionally converts target peaks to query genome using liftOver (if chainFile is provided)
3. If pool=TRUE, pools all target peaks and tests overlap with the pooled set
4. If pool=FALSE, tests overlap with each target peak set separately
5. For permutation testing:
 - Calculates the observed overlap ratio (overlapping peaks / total target peaks)
 - Randomly shuffles target peaks nShuffle times across the genome (preserving chromosome and peak width)
 - Calculates overlap ratio for each shuffled set

- Computes p-value as (number of shuffled ratios $\geq$ observed ratio + 1) / (nShuffle + 1)
6. Adjusts p-values for multiple testing

Direction of the test. The observed quantity is the fraction of *target* peaks covered by the query peaks, $N_OL / \text{length}(\text{targetPeak})$, and the null distribution is generated by shuffling the *target* peaks. The test therefore answers "are the query peaks enriched within the target peaks?". N_OL itself is symmetric (it is the same number in both directions), but the p-value is not: exchanging queryPeak and targetPeak asks the mirrored question and normalises by the other peak set, so the p-values can differ substantially when the two peak sets differ in size. Even for equally sized sets the two null distributions are sampled independently, which leaves small Monte-Carlo differences (see issue #84). Compare N_OL directly if a direction-free measure is needed, increase nShuffle to reduce the Monte-Carlo part, or use symmetric=TRUE for a direction-free p-value (the mirrored direction is then computed as well and combined as $\min(1, 2 \cdot \min(p, p_rev))$, which doubles the number of permutations and raises the smallest reportable p-value to $2 / (nShuffle + 1)$).

Value

A data.frame with one row per target peak comparison, containing:

- qSample: name of the query sample
- tSample: name of the target sample(s)
- qLen: number of peaks in query
- tLen: number of peaks in target (or pooled target if pool=TRUE)
- N_OL: number of overlapping peaks
- pvalue: p-value from permutation test (NA if nShuffle=0)
- p.adjust: adjusted p-value

Author(s)

G Yu

filter.GRanges

Filter GRanges objects using dplyr syntax

Description

The function filters GRanges objects by converting them to a data.frame, applying the dplyr filter operation, and converting back to GRanges. This allows filtering based on any metadata column or standard GRanges columns (seqnames, start, end, width, strand).

Usage

```
## S3 method for class 'GRanges'
filter(.data, ..., .by = NULL, .preserve = FALSE)
```

Arguments

<code>.data</code>	GRanges object to filter
<code>...</code>	logical expressions indicating which ranges to keep. Can use any column names from the GRanges metadata or standard GRanges columns (seqnames, start, end, width, strand). Multiple conditions are combined with AND logic
<code>.by</code>	optional grouping variable (for dplyr compatibility). Currently not used but included for API consistency
<code>.preserve</code>	logical, whether to preserve grouping structure (for dplyr compatibility). Currently not used but included for API consistency. Default is FALSE

Details

This function extends `dplyr::filter()` to work with GRanges objects, allowing users to filter genomic ranges using familiar dplyr syntax while preserving the GRanges structure and all metadata columns.

The function performs the following steps:

1. Converts GRanges to data.frame using `as.data.frame()`
2. Applies `dplyr::filter()` with the provided conditions
3. Removes unused factor levels using `droplevels()`
4. Converts back to GRanges using `makeGRangesFromDataFrame()` with `keep.extra.columns=TRUE`

All metadata columns are preserved in the output. The function supports all standard dplyr filter operations including logical operators, comparisons, and functions like `is.na()`, `between()`, etc.

Value

GRanges object containing only the filtered ranges that satisfy the conditions. All metadata columns are preserved, and unused factor levels are dropped

Examples

```
## Not run:
## Filter peaks by annotation
filtered_peaks <- peak.gr %>% filter(annotation == "Promoter")

## Filter by distance to TSS
near_peaks <- peak.gr %>% filter(abs(distanceToTSS) < 5000)

## Multiple conditions (AND logic)
specific_peaks <- peak.gr %>%
  filter(annotation == "Promoter", abs(distanceToTSS) < 3000)

## Filter by chromosome
chr1_peaks <- peak.gr %>% filter(seqnames == "chr1")

## Complex conditions
complex_filter <- peak.gr %>%
```

```

    filter(annotation %in% c("Promoter", "5' UTR"),
           !is.na(geneId),
           width > 200)

## End(Not run)

```

getAnnoStat

Get annotation statistics from csAnno object

Description

The function returns the annoStat slot, which contains a summary of how peaks are distributed across different genomic feature categories (Promoter, Exon, Intron, UTR, Intergenic, etc.) with percentages.

Usage

```
getAnnoStat(x)
```

Arguments

x csAnno object

Details

This function extracts the annotation statistics data frame from a csAnno object.

Value

A data.frame with columns:

- Feature: genomic feature category (Promoter, 5' UTR, 3' UTR, Exon, Intron, Downstream, Intergenic, etc.)
- Frequency: percentage of peaks in each category

Only available if hasGenomicAnnotation=TRUE

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

See Also

[plotAnnoBar](#) for visualizing these statistics

Examples

```
## Not run:
stats <- getAnnoStat(peakAnno)
## View annotation distribution
print(stats)

## End(Not run)
```

getBioRegion

Prepare a biological region of selected feature from TxDb

Description

The function extracts genomic regions based on feature type and region type. It supports three region types:

- **start_site**: Region around the transcription start site (TSS). For a transcript at chr1:1000-1400, with upstream=100 and downstream=100, this extracts chr1:900-1100 (TSS +/- 100bp)
- **end_site**: Region around the transcription termination site (TTS). For a transcript at chr1:1000-1400, with upstream=100 and downstream=100, this extracts chr1:1300-1500 (TTS +/- 100bp)
- **body**: The full feature body. For a transcript at chr1:1000-1400, this extracts chr1:1000-1400 (the entire transcript)

Usage

```
getBioRegion(
  TxDb = NULL,
  upstream = 1000,
  downstream = 1000,
  by = "gene",
  type = "start_site"
)
```

Arguments

TxDb	TxDb or EnsDb annotation object. If NULL, will attempt to determine from the data
upstream	numeric, distance in base pairs to extend upstream from start site or end site. For body type, this can extend the 5' flank. Default is 1000
downstream	numeric, distance in base pairs to extend downstream from start site or end site. For body type, this can extend the 3' flank. Default is 1000
by	character, one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', 'UTR'. Determines which feature type to extract. Default is "gene"
type	character, one of "start_site", "end_site", "body". Determines which part of the feature to extract. Default is "start_site"

Details

This function extracts biological regions (promoters, gene bodies, UTRs, etc.) from a TxDb or EnsDb annotation object. It provides a unified interface for extracting different types of genomic regions.

The function supports multiple feature types:

- gene: Gene-level regions
- transcript: Transcript-level regions
- exon: Exon regions
- intron: Intron regions
- 3UTR, 5UTR, UTR: UTR regions

For start_site and end_site types, the function creates regions extending upstream and downstream from the site. For body type, it returns the full feature body. The function handles strand information correctly (upstream/downstream are relative to the feature's strand).

Value

GRanges object containing the extracted regions with attributes:

- type: The region type ("start_site", "end_site", or "body")
- by: The feature type used
- label: Generated label for the regions
- upstream: Upstream distance (for start_site/end_site)
- downstream: Downstream distance (for start_site/end_site)

Author(s)

Guangchuang Yu, Ming L

See Also

[getPromoters](#) for a convenience function to get promoters, [makeBioRegionFromGranges](#) for creating regions from custom GRanges

getGeneAnno

Get gene annotation from annotation database

Description

The function queries an annotation database to retrieve gene information such as gene symbols, full gene names, Ensembl IDs, and other identifiers. It handles both Entrez Gene IDs and Ensembl Gene IDs, and returns a data frame with the requested annotation columns.

Usage

```
getGeneAnno(annoDb, geneID, type, columns)
```

Arguments

annoDb	character, name of the annotation database package (e.g., "org.Hs.eg.db" for human). The package must be installed and loaded
geneID	character vector of gene IDs to query. Can be Entrez Gene IDs or Ensembl Gene IDs depending on the type parameter
type	character, gene ID type. Must be one of "Entrez Gene ID" or "Ensembl gene ID" / "Ensembl Gene ID"
columns	character vector, names of columns to retrieve from the annotation database. Common values include "SYMBOL", "GENENAME", "ENSEMBL", "ENTREZID", "GO", "PATH", etc. See the annotation package documentation for available columns

Details

This function retrieves gene annotations (symbol, gene name, etc.) from annotation databases (e.g., org.Hs.eg.db) based on gene IDs.

The function performs the following steps:

1. Validates and converts the gene ID type to the appropriate keytype for the annotation database (ENTREZID or ENSEMBL)
2. Removes version numbers from Ensembl IDs (e.g., "ENSG00000123456.1" becomes "ENSG00000123456")
3. Queries the annotation database using `AnnotationDbi::select()`
4. Handles duplicate gene IDs by keeping only the first occurrence
5. Returns a data frame with rows matching the input gene IDs

Value

A data.frame with:

- Rows corresponding to the input gene IDs (in the same order)
- Columns as specified in the columns parameter, plus the keytype column (ENTREZID or ENSEMBL)
- NA values for gene IDs that could not be found in the database

Returns NA if the ID type is not supported or if the database query fails

Author(s)

G Yu

getGEOgenomeVersion	<i>getGEOgenomeVersion</i>
---------------------	----------------------------

Description

get genome version statistics collecting from GEO ChIPseq data

Usage

```
getGEOgenomeVersion()
```

Value

data.frame

Author(s)

G Yu

getGEOInfo	<i>getGEOInfo</i>
------------	-------------------

Description

get subset of GEO information by genome version keyword

Usage

```
getGEOInfo(genome, simplify = TRUE)
```

Arguments

genome	genome version
simplify	simplify result or not

Value

data.frame

Author(s)

G Yu

getGEOspecies

getGEOspecies

Description

accessing species statistics collecting from GEO database

Usage

```
getGEOspecies()
```

Value

data.frame

Author(s)

G Yu

getNearestFeatureIndicesAndDistances

Get index of nearest features to peaks and calculate distances

Description

The function finds the closest feature to each peak by considering both upstream and downstream directions. It calculates distances from both the peak start and peak end to the feature's TSS, then selects the feature with the minimum absolute distance. The function handles overlaps between peaks and features, with different behaviors depending on the overlap parameter.

Usage

```
getNearestFeatureIndicesAndDistances(
  peaks,
  features,
  sameStrand = FALSE,
  ignoreOverlap = FALSE,
  ignoreUpstream = FALSE,
  ignoreDownstream = FALSE,
  overlap = "TSS"
)
```

Arguments

peaks	GRanges object containing genomic ranges of peaks
features	GRanges object containing genomic features (genes or transcripts) to search for nearest features. Typically obtained from <code>getGene(TxDb)</code> or similar functions
sameStrand	logical, whether to only consider features on the same strand as the peak, both when finding the nearest feature and when detecting overlaps with it. If FALSE, searches both strands. Peaks with ambiguous strand ('*') are compatible with features on any strand. Default is FALSE
ignoreOverlap	logical, whether to ignore overlaps between peaks and features when finding the nearest feature. If FALSE, overlapping features will be prioritized and assigned distance=0 (for TSS overlaps) or calculated distance (for full feature overlaps). Default is FALSE
ignoreUpstream	logical, if TRUE, only considers features downstream (3' end) of the peak. This restricts annotation to genes that come after the peak. Default is FALSE
ignoreDownstream	logical, if TRUE, only considers features upstream (5' end) of the peak. This restricts annotation to genes that come before the peak. Default is FALSE
overlap	character, one of "TSS" or "all". Determines how overlaps are detected and handled: • "TSS": Only considers overlaps with the TSS point (after features are resized to width=1). Overlapping peaks get distance=0 • "all": Considers overlaps with any part of the full feature range (before resizing). For overlapping peaks, calculates distance from the closer peak end (start or end) to the feature's TSS Default is "TSS"

Details

This function identifies the nearest gene/transcript feature to each peak and calculates the distance from the peak to the feature's transcription start site (TSS).

The function performs the following steps:

1. Resizes features to width=1 (TSS points) for efficient distance calculation
2. Finds nearest upstream features using `follow()` and nearest downstream features using `precede()`
3. Calculates distances from both peak start and peak end to each feature's TSS
4. Selects the feature with the minimum absolute distance (or enforces direction based on `ignoreUpstream/ignoreDownstream`)
5. If overlaps are allowed (`ignoreOverlap=FALSE`), identifies overlapping features and assigns them with distance=0 (for `overlap="TSS"`) or calculates distance from the closer peak end (for `overlap="all"`)
6. Returns indices, distances, and filtered peaks (excluding peaks with no nearest feature)

Distance calculation is strand-aware: positive values indicate downstream, negative values indicate upstream. Overlapping peaks get distance=0 if overlap is "TSS".

Value

A list with three components:

- **index**: Integer vector of feature indices (1-based) for the nearest feature to each peak. Length equals the number of peaks with valid nearest features
- **distance**: Numeric vector of distances from each peak to the TSS of its nearest feature. Positive values indicate downstream, negative values indicate upstream, and 0 indicates overlap. Length equals the number of peaks with valid nearest features
- **peak**: GRanges object containing only the peaks that have valid nearest features (peaks with no nearest feature in either direction are excluded)

Author(s)

G Yu

getPromoters

Prepare promoter regions from TxDb

Description

The function extracts promoter regions defined as regions around transcription start sites (TSS). For each gene or transcript, it creates a region extending upstream and downstream from the TSS. The function calls `getBioRegion()` with `type="start_site"`.

Usage

```
getPromoters(TxDb = NULL, upstream = 1000, downstream = 1000, by = "gene")
```

Arguments

<code>TxDb</code>	TxDb or EnsDb annotation object. If NULL, will attempt to determine from the data
<code>upstream</code>	numeric, distance in base pairs to extend upstream from TSS. Default is 1000
<code>downstream</code>	numeric, distance in base pairs to extend downstream from TSS. Default is 1000
<code>by</code>	character, one of "gene" or "transcript". Determines whether to extract promoters at gene level or transcript level. Default is "gene"

Details

This function extracts promoter regions (TSS regions) from a TxDb or EnsDb annotation object. It is a convenience wrapper around `getBioRegion()` for extracting promoter regions.

Value

GRanges object containing promoter regions with attributes:

- type: "start_site"
- by: the value of the by parameter
- label: generated label for the regions
- upstream: upstream distance
- downstream: downstream distance

See Also

[getBioRegion](#) for more flexible region extraction

getSampleFiles

Get sample peak files included in the package

Description

The function scans the package's extdata directory for sample peak files and returns their paths. The files are named based on the protein/condition they represent (extracted from the filename).

Usage

```
getSampleFiles()
```

Details

This function returns a list of sample ChIP-seq peak files included in the package for demonstration and testing purposes.

Value

A named list of file paths. Names are extracted from filenames (typically protein/condition names), and values are full file paths to the sample BED files

Author(s)

G Yu

getTagMatrix

Calculate tag matrix for peak coverage visualization

Description

The function computes peak coverage across a set of genomic regions (windows). It supports two methods:

- **Direct method** (when nbin=NULL): For regions of equal size (e.g., promoters), calculates coverage at each base pair position
- **Binning method** (when nbin is specified): For regions of variable size (e.g., gene bodies), divides each region into a fixed number of bins and calculates average coverage per bin

Usage

```
getTagMatrix(
  peak,
  upstream,
  downstream,
  windows,
  type,
  by,
  Txdb = NULL,
  weightCol = NULL,
  nbin = NULL,
  verbose = TRUE,
  ignore_strand = FALSE
)
```

Arguments

peak	peak file (BED format) or GRanges object containing ChIP-seq peaks
upstream	numeric or rel() object, distance to extend upstream. For type='body', can extend 5' flank. For type='start_site'/'end_site', only used when windows is missing. Can be NULL for body regions
downstream	numeric or rel() object, distance to extend downstream. For type='body', can extend 3' flank. For type='start_site'/'end_site', only used when windows is missing. Can be NULL for body regions
windows	GRanges object containing regions of interest. Should be created by getPromoters(), getBioRegion(), or makeBioRegionFromGranges(). If missing, will be created from Txdb and other parameters
type	character, one of "start_site", "end_site", "body". Required if windows is missing
by	character, one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', or user-specified label. Required if windows is missing

TxDb	TxDb/EnsDb object or GRanges object. If TxDb, used to create windows. If GRanges, treated as custom regions and passed to makeBioRegionFromGranges(). Required if windows is missing
weightCol	character, name of a metadata column in the peak GRanges to use as weights for coverage calculation. If NULL, all peaks have equal weight. Default is NULL
nbin	integer, number of bins for binning method. Required for type='body', optional for other types. If NULL, uses direct method (requires equal-width windows). Default is NULL
verbose	logical, whether to print progress messages. Default is TRUE
ignore_strand	logical, whether to ignore strand information when calculating coverage. If FALSE, negative strand regions are reverse-complemented. Default is FALSE

Details

This function calculates a tag matrix representing peak coverage across genomic regions (e.g., promoters, gene bodies). The matrix can be used for visualization with functions like plotPeakProf() or plotAvgProf().

The function can work in two modes:

1. **With pre-made windows:** Provide a windows GRanges object created by getPromoters(), getBioRegion(), or makeBioRegionFromGranges()
2. **Without windows:** Provide TxDb and region parameters (type, by, upstream, downstream), and the function will create windows automatically

For upstream and downstream parameters:

- If windows is provided: For type='body', these extend the flanks; for type='start_site'/'end_site', they are ignored (taken from window attributes)
- If windows is missing: These define the region size for type='start_site'/'end_site', or extend flanks for type='body'

The weightCol parameter allows weighting peaks by a metadata column (e.g., peak score or signal value), useful for signal-weighted coverage.

Value

A matrix (tagMatrix) with:

- Rows: One per region (window)
- Columns: One per position/bin
- Values: Peak coverage/signal at each position

The matrix has attributes:

- upstream: Upstream distance
- downstream: Downstream distance
- type: Region type
- label: Region label
- is.binning: Whether binning method was used

See Also

[getPromoters](#), [getBioRegion](#), [makeBioRegionFromGranges](#) for creating windows, [plotPeakProf](#), [plotAvgProf](#) for visualization

getTagMatrix.binning.internal

Calculate tag matrix using binning method (internal function)

Description

The function divides each region into a fixed number of bins and calculates average coverage per bin. This allows handling regions of variable size (e.g., gene bodies of different lengths) by normalizing them to the same number of bins. The approach is similar to deeptools computeMatrix (<https://deeptools.readthedocs.io/en/develop/content/tools/computeMatrix.html>).

Usage

```
getTagMatrix.binning.internal(
  peak,
  weightCol = NULL,
  windows,
  nbin = 800,
  upstream = NULL,
  downstream = NULL,
  ignore_strand = FALSE
)
```

Arguments

peak	peak file (BED format) or GRanges object containing ChIP-seq peaks
weightCol	character, name of a metadata column to use as weights. If NULL, all peaks have equal weight. Default is NULL
windows	GRanges object containing regions. Regions can have variable sizes (e.g., gene bodies of different lengths)
nbin	integer, number of bins to divide each region into. Should not exceed the minimum region length. Default is 800
upstream	numeric, rel() object, or NULL. Distance to extend upstream. For type='body', can extend 5' flank. If rel(), extends by a fraction of region width. Default is NULL
downstream	numeric, rel() object, or NULL. Distance to extend downstream. For type='body', can extend 3' flank. If rel(), extends by a fraction of region width. Default is NULL
ignore_strand	logical, whether to ignore strand information. If FALSE, negative strand regions are reverse-complemented. Default is FALSE

Details

This is an internal function that calculates peak coverage using a binning approach, inspired by `deeptools computeMatrix`. It is called by `getTagMatrix()` when `nbin` is specified.

The function performs the following steps:

1. Loads peaks and calculates coverage (optionally weighted)
2. Filters windows to those within chromosome boundaries
3. Optionally extends windows by upstream and downstream (can be relative values using `rel()`)
4. For each window, divides it into `nbin` equal-sized bins
5. Calculates average coverage per bin
6. Creates a matrix with one row per window and one column per bin
7. Reverse-complements negative strand regions (if `ignore_strand=FALSE`)
8. Filters out windows shorter than the minimum required length

The binning method is essential for visualizing coverage across variable-length regions (e.g., gene bodies), as it normalizes all regions to the same number of data points.

Value

A matrix with:

- Rows: One per window (regions meeting minimum length requirement)
- Columns: One per bin (`nbin` columns)
- Values: Average coverage/signal per bin

info

Information Datasets

Description

Pre-calculated datasets included in the `ChIPseeker` package:

- `ucsc_release`: UCSC genome version information
- `gsminfo`: GEO Sample (GSM) information for ChIP-seq datasets
- `tagMatrixList`: Pre-calculated tag matrices for example datasets

makeBioRegionFromGranges

Make biological regions from user-provided GRanges object

Description

The function extracts regions from a custom GRanges object, similar to getBioRegion() but working with user-defined features instead of TxDb annotations. This is useful for features like enhancers, insulators, or other regulatory elements that are not part of standard gene annotations.

Usage

```
makeBioRegionFromGranges(gr, by, type, upstream = 1000, downstream = 1000)
```

Arguments

gr	GRanges object containing regions of interest (e.g., enhancers, insulators, custom regulatory elements)
by	character, user-specified label for the feature type (e.g., "gene", "insulator", "enhancer"). Used for generating region labels. Cannot be omitted
type	character, one of "start_site", "end_site", "body". Determines which part of each range to extract. Cannot be omitted
upstream	numeric or NULL, distance in base pairs to extend upstream from start site or end site. Can be NULL if type == 'body'. Default is 1000
downstream	numeric or NULL, distance in base pairs to extend downstream from start site or end site. Can be NULL if type == 'body'. Default is 1000

Details

This function creates biological regions from a user-provided GRanges object, allowing extraction of regions around start sites, end sites, or full bodies for custom genomic features not available in TxDb objects (e.g., enhancers, insulators).

The function supports three region types:

- **start_site**: Region around the start of each range. For an enhancer at chr1:1000-1400 with upstream=100 and downstream=100, extracts chr1:900-1100 (start +/- 100bp, strand-aware)
- **end_site**: Region around the end of each range. For an enhancer at chr1:1000-1400 with upstream=100 and downstream=100, extracts chr1:1300-1500 (end +/- 100bp, strand-aware)
- **body**: The full range body. For an enhancer at chr1:1000-1400, extracts chr1:1000-1400. For this type, upstream and downstream can be NULL

The function handles strand information correctly: for negative strand features, "upstream" and "downstream" are reversed relative to the genomic coordinates.

Value

GRanges object containing the extracted regions with attributes:

- type: The region type
- by: The feature type label
- label: Generated label for the regions
- upstream: Upstream distance (for start_site/end_site)
- downstream: Downstream distance (for start_site/end_site)

See Also

[getBioRegion](#) for extracting regions from TxDb objects

mutate.GRanges

Mutate GRanges objects using dplyr syntax

Description

The function mutates GRanges objects by converting them to a data.frame, applying the dplyr mutate operation, and converting back to GRanges. This allows creating new columns or modifying existing metadata columns based on expressions involving other columns.

Usage

```
## S3 method for class 'GRanges'
mutate(
  .data,
  ...,
  .by = NULL,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)
```

Arguments

<code>.data</code>	GRanges object to mutate
<code>...</code>	name-value pairs of expressions. Use name = value to add new columns or modify existing ones. The name becomes the column name, and the value is an expression that can reference other columns
<code>.by</code>	optional grouping variable (for dplyr compatibility). Currently not used but included for API consistency
<code>.keep</code>	character, which columns to keep. Options: "all" (keep all columns), "used" (keep only columns used in expressions), "unused" (keep only columns not used in expressions), "none" (keep only new columns). Default is "all"

`.before`, `.after` column position specifiers. Place new columns before or after the specified column. Cannot use both. If NULL, new columns are appended at the end. Default is NULL for both

Details

This function extends `dplyr::mutate()` to work with `GRanges` objects, allowing users to add new metadata columns or modify existing ones using familiar `dplyr` syntax while preserving the `GRanges` structure.

The function performs the following steps:

1. Converts `GRanges` to `data.frame` using `as.data.frame()`
2. Validates that `.before` and `.after` are not both specified
3. Applies `dplyr::mutate()` with the provided expressions, handling column positioning if `.before` or `.after` is specified
4. Converts back to `GRanges` using `makeGRangesFromDataFrame()` with `keep.extra.columns=TRUE`

All genomic range information (seqnames, ranges, strand) is preserved. The function supports all standard `dplyr` mutate operations including arithmetic, logical operations, and functions.

Value

`GRanges` object with modified or new metadata columns. All genomic range information is preserved. The order and position of new columns follows the `.before` or `.after` specifications

Examples

```
## Not run:
## Add a new column based on existing data
peak.gr <- peak.gr %>% mutate(logScore = log10(score))

## Modify existing column
peak.gr <- peak.gr %>% mutate(distance = abs(distanceToTSS))

## Create multiple columns
peak.gr <- peak.gr %>%
  mutate(
    distance_kb = abs(distanceToTSS) / 1000,
    is_promoter = annotation == "Promoter",
    category = ifelse(abs(distanceToTSS) < 3000, "near", "far")
  )

## Place new column before specific column
peak.gr <- peak.gr %>%
  mutate(new_col = 1, .before = annotation)

## Keep only used columns
peak.gr <- peak.gr %>%
  mutate(result = score * 2, .keep = "used")

## End(Not run)
```

`overlap`*Calculate overlap matrix for sets*

Description

The function computes the number of elements in each possible intersection of the input sets. It handles all combinations of set intersections using an inclusion-exclusion principle approach.

Usage

```
overlap(Sets)
```

Arguments

Sets	list of objects (e.g., GRanges objects, character vectors, etc.) to calculate overlaps for. Each element should support <code>intersect()</code> and <code>length()</code> methods
------	--

Details

This function calculates the overlap matrix for multiple sets, which is useful for creating Venn diagrams and understanding set relationships.

The function:

1. Generates all possible combinations of set inclusion/exclusion
2. For each combination, calculates the intersection size
3. Uses inclusion-exclusion principle to handle overlapping intersections
4. Returns a data frame with binary indicators for each set and the intersection size

The function is generic and works with any objects that support `intersect()` and `length()` methods, including GRanges objects.

Value

A data.frame with:

- One column per set (named after the sets)
- Binary values (0 or 1) indicating which sets are included in each intersection
- A "Weight" column containing the number of elements in each intersection

Rows represent all possible combinations of set intersections

Author(s)

G Yu

peak_Profile_Heatmap *Plot combined peak heatmap and average profile*

Description

The function generates a two-panel figure:

- Top panel: Heatmap showing signal intensity for each feature (row) across genomic positions (columns)
- Bottom panel: Average profile line showing the mean signal across all features at each position

This combination allows users to see both individual feature patterns and the overall trend simultaneously.

Usage

```
peak_Profile_Heatmap(
  peak,
  weightCol = NULL,
  TxDb = NULL,
  upstream = 1000,
  downstream = 1000,
  xlab = "",
  ylab = "",
  title = NULL,
  palette = NULL,
  verbose = TRUE,
  by = "gene",
  type = "start_site",
  nbin = NULL,
  ignore_strand = FALSE,
  windows_name = NULL,
  ncol = NULL,
  nrow = NULL,
  facet_label_text_size = 12,
  conf,
  facet = "row",
  free_y = TRUE,
  height_proportion = 4
)
```

Arguments

peak	peak file (BED format) or GRanges object, or a named list of peak files/GRanges objects for comparing multiple datasets
weightCol	character, name of metadata column to use as weights. If NULL, all peaks have equal weight. Default is NULL

TxDb	TxDb or EnsDb annotation object, or a GRanges object containing custom genomic features
upstream	numeric, upstream distance in base pairs. Default is 1000
downstream	numeric, downstream distance in base pairs. Default is 1000
xlab	character, x-axis label. Default is "" (empty)
ylab	character, y-axis label. Default is "" (empty)
title	character, plot title. If NULL, no title is shown. Default is NULL
palette	character, color palette name from RColorBrewer for the heatmap. See scale_colour_brewer for options. Default is NULL
verbose	logical, whether to print progress messages. Default is TRUE
by	character, feature type. With TxDb: one of "gene", "transcript", "exon", "intron", "3UTR", "5UTR", "UTR". Default is "gene"
type	character, one of "start_site", "end_site", or "body". Default is "start_site"
nbin	integer, number of bins for binning mode. If NULL, uses absolute positioning. Default is NULL
ignore_strand	logical, whether to ignore strand information. Default is FALSE
windows_name	character vector, names for windows (for multi-window plots). Default is NULL
ncol	integer, number of columns for arranging multiple plots. Default is NULL
nrow	integer, number of rows for arranging multiple plots. Default is NULL
facet_label_text_size	numeric, size of facet label text. Default is 12
conf	numeric, confidence level for confidence intervals in the profile plot (e.g., 0.95). If NA or missing, no confidence intervals are shown
facet	character, one of "none", "row", or "column" for faceting the profile plot. Default is "none"
free_y	logical, if TRUE, y-axis scales independently per facet in the profile plot. Default is TRUE
height_proportion	numeric, proportion of height allocated to profile plot vs heatmap. Values between 0 and 1, where 0.5 means equal heights. Default is 0.5

Details

This function creates a combined visualization showing both a heatmap of individual features and an average profile line plot in a single figure, providing both detailed and summary views of peak distributions.

peakHeatmap

*Plot heatmap of peaks around genomic features***Description**

The function generates tag matrices from peaks and visualizes them as heatmaps. It supports both absolute positioning and binning modes. Features are typically sorted by signal strength, allowing identification of features with strong peak enrichment.

Usage

```
peakHeatmap(
  peak,
  weightCol = NULL,
  TxDb = NULL,
  upstream = 1000,
  downstream = 1000,
  xlab = "",
  ylab = "",
  title = NULL,
  palette = NULL,
  verbose = TRUE,
  by = "gene",
  type = "start_site",
  nbin = NULL,
  ignore_strand = FALSE,
  windows,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

peak	peak file (BED format) or GRanges object, or a named list of peak files/GRanges objects for comparing multiple datasets
weightCol	character, name of metadata column to use as weights. If NULL, all peaks have equal weight. Default is NULL
TxDb	TxDb or EnsDb annotation object, or a GRanges object containing custom genomic features
upstream	numeric, upstream distance in base pairs. Default is 1000
downstream	numeric, downstream distance in base pairs. Default is 1000
xlab	character, x-axis label. Default is "" (empty)
ylab	character, y-axis label. Default is "" (empty)
title	character, plot title. If NULL, no title is shown. Default is NULL

palette	character, color palette name from RColorBrewer. If NULL, uses default. See scale_colour_brewer for options. Default is NULL
verbose	logical, whether to print progress messages. Default is TRUE
by	character, feature type. With TxDb: one of "gene", "transcript", "exon", "intron", "3UTR", "5UTR", "UTR". Default is "gene"
type	character, one of "start_site", "end_site", or "body". Default is "start_site"
nbin	integer, number of bins for binning mode. If NULL, uses absolute positioning. Default is NULL
ignore_strand	logical, whether to ignore strand information. Default is FALSE
windows	GRanges object, custom genomic regions to use instead of generating from TxDb. If missing, regions are generated from TxDb. Default is missing
ncol	integer, number of columns for arranging multiple heatmaps. Default is NULL
nrow	integer, number of rows for arranging multiple heatmaps. Default is NULL

Details

This function creates a heatmap showing the distribution of ChIP-seq peaks around genomic features (e.g., TSS, gene body). Each row represents a feature and each column represents a genomic position.

The function:

1. Generates tag matrices from peaks using `getTagMatrix()`
2. Optionally uses custom windows (if provided) or generates them from TxDb based on `by` and `type`
3. Creates heatmap visualization with color-coded signal intensities
4. Supports single or multiple peak sets (list input)

Value

A `ggplot2` heatmap object showing peak signals across features and positions. For multiple peak sets, returns a grid of heatmaps

Author(s)

G Yu

peakHeatmap_multiple_Sets

Plot heatmap of peaks aligned to multiple sets of regions

Description

The function extends `peakHeatmap()` to support multiple window sets. It generates tag matrices for each window set and creates a faceted heatmap visualization, with each facet showing peaks aligned to a different set of regions. This is useful for comparing peak patterns across different feature types (e.g., promoters vs enhancers) or conditions.

Usage

```
peakHeatmap_multiple_Sets(
  peak,
  weightCol = NULL,
  TxDb = NULL,
  upstream = 1000,
  downstream = 1000,
  xlab = "",
  ylab = "",
  title = NULL,
  palette = NULL,
  verbose = TRUE,
  by = "gene",
  type = "start_site",
  nbin = NULL,
  ignore_strand = FALSE,
  windows_name = NULL,
  ncol = NULL,
  nrow = NULL,
  facet_label_text_size = 12
)
```

Arguments

peak	peak file (BED format) or GRanges object, or a named list of peak files/GRanges objects for comparing multiple datasets
weightCol	character, name of metadata column to use as weights. If NULL, all peaks have equal weight. Default is NULL
TxDb	TxDb or EnsDb annotation object, or a named list of GRanges objects containing custom genomic features. If a list, names are used as window names
upstream	numeric, upstream distance in base pairs. Must be the same for all windows. Default is 1000
downstream	numeric, downstream distance in base pairs. Must be the same for all windows. Default is 1000

xlab	character, x-axis label. Default is "" (empty)
ylab	character, y-axis label. Default is "" (empty)
title	character, plot title. If NULL, no title is shown. Default is NULL
palette	character, color palette name from RColorBrewer. If NULL, uses default. Default is NULL
verbose	logical, whether to print progress messages. Default is TRUE
by	character vector, feature types for each window. With TxDb: one of "gene", "transcript", "exon", "intron", "3UTR", "5UTR", "UTR". Length must match number of windows. Default is "gene"
type	character, one of "start_site", "end_site", or "body". Must be the same for all windows. Default is "start_site"
nbin	integer, number of bins for binning mode. If NULL, uses absolute positioning. Default is NULL
ignore_strand	logical, whether to ignore strand information. Default is FALSE
windows_name	character vector, names for each window (displayed as facet labels). If NULL and TxDb is a named list, uses names from TxDb. Default is NULL
ncol	integer, number of columns for arranging heatmaps. Default is NULL (auto)
nrow	integer, number of rows for arranging heatmaps. Default is NULL (auto)
facet_label_text_size	numeric, size of facet label text. Default is 12

Details

This function creates heatmaps showing peak distributions across multiple sets of genomic regions (windows), allowing comparison of peak patterns across different feature types or conditions.

The function:

1. Generates tag matrices for each window set (specified via by or custom TxDb GRanges)
2. Creates heatmaps for each window set
3. Arranges them in a grid with faceting
4. Supports both absolute positioning and binning modes

All windows must have the same type (start_site, end_site, or body) and the same upstream and downstream parameters.

Value

figure

plotAnnoBar

Plot bar chart of genomic annotation distribution

Description

The function visualizes the percentage of peaks falling into each genomic annotation category. It creates a horizontal bar chart where each bar represents a category, and the bar length represents the percentage of peaks in that category. The function supports both single csAnno objects and lists of csAnno objects for comparing multiple datasets.

Usage

```
plotAnnoBar(
  x,
  xlab = "",
  ylab = "Percentage(%)",
  title = "Feature Distribution",
  ...
)
```

Arguments

x	csAnno object or list of csAnno objects containing annotated peaks with genomic annotation information
xlab	character, label for x-axis. Default is "" (empty)
ylab	character, label for y-axis. Default is "Percentage(%)"
title	character, plot title. Default is "Feature Distribution"
...	additional parameters passed to the implementation methods

Details

This generic function creates a bar plot showing the distribution of peaks across different genomic annotation categories (Promoter, Exon, Intron, etc.).

The function:

1. Extracts annotation statistics from the csAnno object(s)
2. Calculates percentages for each annotation category
3. Creates a horizontal bar chart with categories on the y-axis and percentages on the x-axis
4. Color-codes each annotation category

Common annotation categories include: Promoter, 5' UTR, 3' UTR, Exon, Intron, Downstream, and Intergenic.

Value

A ggplot2 bar plot object showing the distribution of peaks across genomic annotation categories. For lists of csAnno objects, the plot shows side-by-side bars for comparison

See Also

[plotAnnoPie](#) for pie chart visualization, [annotatePeak](#) for peak annotation

```
plotAnnoBar.data.frame  
      plotAnnoBar.data.frame
```

Description

plot feature distribution based on their chromosome region

Usage

```
plotAnnoBar.data.frame(  
  anno.df,  
  xlab = "",  
  ylab = "Percentage%",  
  title = "Feature Distribution",  
  categoryColumn  
)
```

Arguments

anno.df	annotation stats
xlab	xlab
ylab	ylab
title	plot title
categoryColumn	category column

Details

plot chromosome region features

Value

bar plot that summarize genomic features of peaks

Author(s)

Guangchuang Yu <https://yulab-smu.top>

See Also

[annotatePeak](#) [plotAnnoPie](#)

plotAnnoPie

Plot pie chart of genomic annotation distribution

Description

The function visualizes the percentage of peaks falling into each genomic annotation category using a pie chart. Each slice of the pie represents an annotation category, with the slice size proportional to the percentage of peaks in that category. Labels show the category name and percentage.

Usage

```
plotAnnoPie(
  x,
  ndigit = 2,
  cex = 0.9,
  col = NA,
  legend.position = "rightside",
  pie3D = FALSE,
  radius = 0.8,
  ...
)
```

Arguments

x	csAnno object containing annotated peaks with genomic annotation information
ndigit	integer, number of decimal places to round percentages in labels. Default is 2
cex	numeric, character expansion factor for labels. Larger values make labels bigger. Default is 0.9
col	character vector, colors for pie slices. If NA (default), automatic colors are used
legend.position	character, position of the legend. Options include "rightside", "leftside", "top", "bottom", or specific coordinates. Default is "rightside"
pie3D	logical, whether to create a 3D pie chart. If TRUE, uses pie3D() from the plotrix package. Default is FALSE
radius	numeric, radius of the pie chart. Values between 0 and 1. Default is 0.8
...	additional parameters passed to the pie chart plotting functions

Details

This generic function creates a pie chart showing the distribution of peaks across different genomic annotation categories.

The function:

1. Extracts annotation statistics from the csAnno object
2. Calculates percentages for each annotation category

3. Creates a pie chart with color-coded slices
4. Adds labels showing category names and percentages (rounded to ndigit decimal places)
5. Optionally creates a 3D pie chart for enhanced visualization

Common annotation categories include: Promoter, 5' UTR, 3' UTR, Exon, Intron, Downstream, and Intergenic.

Value

A pie chart plot showing the distribution of peaks across genomic annotation categories. The plot includes labels with category names and percentages, and a legend

See Also

[plotAnnoBar](#) for bar chart visualization, [vennpie](#) for nested pie chart visualization, [annotatePeak](#) for peak annotation

plotAnnoPie.csAnno *plotAnnoPie*

Description

pieplot from peak genomic annotation

Usage

```
plotAnnoPie.csAnno(
  x,
  ndigit = 2,
  cex = 0.8,
  col = NA,
  legend.position = "rightside",
  pie3D = FALSE,
  radius = 0.8,
  ...
)
```

Arguments

x	csAnno object
ndigit	number of digit to round
cex	label cex
col	color
legend.position	topright or other.
pie3D	plot in 3D or not
radius	radius of Pie
...	extra parameter

Value

pie plot of peak genomic feature annotation

Author(s)

Guangchuang Yu <https://yulab-smu.top>

See Also

[annotatePeak](#) [plotAnnoBar](#)

Examples

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfile <- system.file("extdata", "sample_peaks.txt", package="chipseeker")
peakAnno <- annotatePeak(peakfile, TxDb=txdb)
plotAnnoPie(peakAnno)

## End(Not run)
```

plotAvgProf

plot the profile of peaks

Description

The function creates a line plot showing the average signal intensity of peaks across genomic regions. It can plot single or multiple peak sets, with optional confidence intervals (error bars or ribbons) and faceting options for comparing datasets.

Usage

```
plotAvgProf(
  tagMatrix,
  xlim,
  xlab = "Genomic Region (5'→3')",
  ylab = "Peak Count Frequency",
  conf,
  facet = "none",
  free_y = TRUE,
  origin_label = "TSS",
  verbose = TRUE,
  ...
)
```

Arguments

tagMatrix	matrix or list of matrices. Each matrix should have features as rows and genomic positions as columns. Tag matrices are typically obtained from <code>getTagMatrix()</code> . If a list is provided, each element should be a named tag matrix
xlim	numeric vector of length 2, x-axis limits (start, end positions). Should match the number of columns in tagMatrix. Default example: <code>c(-3000, 3000)</code>
xlab	character, x-axis label. Default is "Genomic Region (5'→3')"
ylab	character, y-axis label. Default is "Peak Count Frequency"
conf	numeric, confidence level (e.g., 0.95 for 95 confidence intervals are calculated and displayed as ribbons. If NA or missing, no confidence intervals are shown
facet	character, one of "none", "row", or "column". Controls how multiple datasets are displayed: "none" overlays them, "row" creates row facets, "column" creates column facets. Default is "none"
free_y	logical, if TRUE and faceting is used, y-axis scales independently for each facet. Default is TRUE
origin_label	character, label for the center point (e.g., "TSS", "TES"). A vertical line is drawn at position 0 with this label. Default is "TSS"
verbose	logical, whether to print progress messages. Default is TRUE
...	additional arguments passed to ggplot2 functions

Details

Plot average peak profile

This function plots the average profile of peaks around genomic features (e.g., TSS, gene body) as a line plot, optionally with confidence intervals.

The function:

1. Calculates average signal across all features for each position
2. Optionally calculates confidence intervals (if conf is provided)
3. Creates a line plot with optional ribbon for confidence intervals
4. Supports faceting for multiple datasets (row, column, or none)
5. Allows independent y-axis scaling per facet

For multiple datasets (list input), the function automatically creates separate facets or overlays them based on the facet parameter.

Value

A ggplot2 object showing the average peak profile as a line plot, with optional confidence intervals and faceting

Author(s)

G Yu; Y Yan

plotAvgProf2

plotAvgProf2

Description

plot the profile of peaks that align to flank sequences of TSS

Usage

```
plotAvgProf2(
  peak,
  weightCol = NULL,
  TxDb = NULL,
  upstream = 1000,
  downstream = 1000,
  xlab = "Genomic Region (5'->3')",
  ylab = "Peak Count Frequency",
  conf,
  facet = "none",
  free_y = TRUE,
  verbose = TRUE,
  ignore_strand = FALSE,
  ...
)
```

Arguments

peak	peak file (BED format) or GRanges object, or a list of peak files/GRanges objects for comparing multiple datasets
weightCol	character, name of a metadata column in the GRanges object to use as weights for coverage calculation. If NULL, all peaks have equal weight. Default is NULL
TxDb	TxDb or EnsDb annotation object. Default is NULL
upstream	numeric, upstream distance in base pairs from the start site. Default is 1000
downstream	numeric, downstream distance in base pairs from the start site. Default is 1000
xlab	character, x-axis label. Default is "Genomic Region (5'->3')"
ylab	character, y-axis label. Default is "Peak Count Frequency"
conf	numeric, confidence level (e.g., 0.95 for 95 confidence intervals are displayed. If NA or missing, no confidence intervals are shown
facet	character, one of "none", "row", or "column". Controls how multiple datasets are displayed. Default is "none"
free_y	logical, if TRUE and faceting is used, y-axis scales independently for each facet. Default is TRUE
verbose	logical, whether to print progress messages. Default is TRUE

ignore_strand logical, whether to ignore strand information when calculating coverage. Default is FALSE

... additional arguments passed to ggplot2 functions

Details

This function is the old function of plotPeakProf2. It can only plot the start site region of gene.

Value

ggplot object

Author(s)

G Yu, Ming L

plotDistToTSS	<i>Plot feature distribution based on distances to TSS</i>
---------------	--

Description

The function visualizes where ChIP-seq peaks are located relative to gene TSSs. It categorizes peaks into distance bins (e.g., 0-1kb, 1-3kb, etc.) and displays the percentage of peaks in each category, separately for upstream (5') and downstream (3') regions. The plot uses a diverging bar chart with TSS at the center (0).

Usage

```
plotDistToTSS(
  x,
  distanceColumn = "distanceToTSS",
  xlab = "",
  ylab = "Binding sites (%) (5'->3')",
  title = "Distribution of transcription factor-binding loci relative to TSS",
  ...
)
```

Arguments

x csAnno object or list of csAnno objects containing annotated peaks with distance information

distanceColumn character, name of the column containing distances from peaks to nearest gene TSS. Default is "distanceToTSS"

xlab character, label for x-axis. Default is "" (empty)

ylab character, label for y-axis. Default is "Binding sites (%) (5'->3')"

title character, plot title. Default is "Distribution of transcription factor-binding loci relative to TSS"

... additional parameters passed to the implementation methods, such as:

- `distanceBreaks`: numeric vector of breakpoints for distance categories (default: `c(0, 1000, 3000, 5000, 10000, 100000)`)
- `palette`: color palette name from `RColorBrewer`
- `categoryColumn`: column for grouping multiple datasets

Details

This generic function creates a bar plot showing the distribution of peaks relative to transcription start sites (TSS), categorizing peaks by their distance from the nearest gene's TSS.

The function:

1. Categorizes peaks into distance bins based on `distanceBreaks`
2. Separates upstream (negative distance) and downstream (positive distance) peaks
3. Calculates percentages for each distance category
4. Creates a diverging bar plot with upstream peaks on the left and downstream peaks on the right, with TSS at the center

Value

A `ggplot2` bar plot object showing the distribution of peaks relative to TSS. The plot uses a diverging bar chart format with upstream (5') on the left and downstream (3') on the right, with TSS at the center

See Also

[annotatePeak](#) for peak annotation, [plotAnnoBar](#) for annotation distribution plots

`plotDistToTSS.data.frame`

Plot feature distribution based on distances to TSS

Description

The function visualizes where ChIP-seq peaks are located relative to gene TSSs. It categorizes peaks into distance bins (e.g., 0-1kb, 1-3kb, etc.) and displays the percentage of peaks in each category, separately for upstream (5') and downstream (3') regions. The plot uses a diverging bar chart with TSS at the center (0).

Usage

```
plotDistToTSS.data.frame(
  peakDist,
  distanceColumn = "distanceToTSS",
  distanceBreaks = c(0, 1000, 3000, 5000, 10000, 1e+05),
  palette = NULL,
  xlab = "",
  ylab = "Binding sites (%) (5'→3')",
  title = "Distribution of transcription factor-binding loci relative to TSS",
  categoryColumn = ".id"
)
```

Arguments

peakDist	data.frame containing peak annotation data with a distance column. Typically obtained from <code>as.data.frame(csAnno)</code> or similar
distanceColumn	character, name of the column containing distances from peaks to nearest gene TSS. Default is "distanceToTSS"
distanceBreaks	numeric vector, breakpoints for distance categories in base pairs. The function automatically adds 0 and Inf if not present. Default is <code>c(0, 1000, 3000, 5000, 10000, 100000)</code>
palette	character, color palette name from <code>RColorBrewer</code> (e.g., "Set1", "Dark2") or NULL for default colors. Run <code>RColorBrewer::display.brewer.all()</code> to see available palettes. Default is NULL
xlab	character, label for x-axis. Default is "" (empty)
ylab	character, label for y-axis. Default is "Binding sites (%) (5'→3')"
title	character, plot title. Default is "Distribution of transcription factor-binding loci relative to TSS"
categoryColumn	character or numeric, column name or index for grouping multiple datasets. If 1 or ".id", treats as single dataset. Default is ".id"

Details

This function creates a bar plot showing the distribution of peaks relative to transcription start sites (TSS), categorizing peaks by their distance from the nearest gene's TSS.

The function performs the following steps:

1. Categorizes peaks into distance bins based on `distanceBreaks`
2. Separates upstream (negative distance) and downstream (positive distance) peaks
3. Calculates percentages for each distance category
4. Creates a diverging bar plot with:
 - Upstream peaks shown as negative bars (left side)
 - Downstream peaks shown as positive bars (right side)
 - TSS marked at the center (0)
 - Distance categories color-coded
5. Supports grouping by a category column for comparing multiple datasets

Value

A ggplot2 bar plot object showing the distribution of peaks relative to TSS. The plot uses a diverging bar chart format with upstream (5') on the left and downstream (3') on the right, with TSS at the center

Author(s)

Guangchuang Yu <https://guangchuangyu.github.io>

See Also

[annotatePeak](#)

Examples

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfile <- system.file("extdata", "sample_peaks.txt", package="ChIPseeker")
peakAnno <- annotatePeak(peakfile, TxDb=txdb)
plotDistToTSS(peakAnno)

## End(Not run)
```

plotPeakProf

plotPeakProf_MultiWindows

Description

The function is a flexible entry point for plotting peak profiles. It can work with either pre-computed tag matrices or raw peak files/GRanges objects. When provided with peaks, it automatically generates tag matrices and plots them. The function routes to different plotting functions based on the input type and parameters.

Usage

```
plotPeakProf(
  tagMatrix = NULL,
  peak,
  upstream,
  downstream,
  conf,
  by,
  type,
  windows_name = NULL,
  weightCol = NULL,
  TxDb = NULL,
  xlab = "Genomic Region (5'->3')",
```

```

    ylab = "Peak Count Frequency",
    facet = "row",
    free_y = TRUE,
    verbose = TRUE,
    nbin = NULL,
    ignore_strand = FALSE,
    ...
)

```

Arguments

tagMatrix	matrix or list of matrices, optional. Pre-computed tag matrix (from <code>getTagMatrix()</code>). If provided, peaks are not required. If NULL, tag matrix will be generated from peaks. Default is NULL
peak	peak file (BED format) or GRanges object, or a list of peak files/GRanges objects for comparing multiple datasets. Required if tagMatrix is NULL
upstream	numeric, upstream distance in base pairs. Interpretation depends on type: for "start_site"/"end_site", it's the distance upstream of the site; for "body", it's the extension beyond the feature body. Can be NULL if using binning mode. Default varies by function
downstream	numeric, downstream distance in base pairs. Interpretation depends on type: for "start_site"/"end_site", it's the distance downstream of the site; for "body", it's the extension beyond the feature body. Can be NULL if using binning mode. Default varies by function
conf	numeric, confidence level (e.g., 0.95 for 95 confidence intervals are calculated and displayed as ribbons around the average profile. If NA or missing, no confidence intervals are shown. Default varies by function
by	character vector, feature(s) of interest. If using TxDb: one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', 'UTR'. If using self-made GRanges: can be any label (used only for plot labels). For plotPeakProf_MultiWindows, can be a vector of length > 1 to compare multiple windows
type	character, one of "start_site", "end_site", or "body". Determines which part of the feature to plot: transcription start site, transcription end site, or the entire feature body. For plotPeakProf_MultiWindows, must be a single value (all windows must have the same type)
windows_name	character vector, optional names for each window (used in plotPeakProf_MultiWindows). These names appear as labels in the plot. If NULL and TxDb is a named list, uses names from TxDb. If NULL and by is provided, uses by values. Default is NULL
weightCol	character, name of a metadata column in the GRanges object to use as weights for coverage calculation. If NULL, all peaks have equal weight. Default is NULL
TxDb	TxDb or EnsDb annotation object, or a list of GRanges objects containing user-defined genomic features. If a list of GRanges is provided, each element should correspond to a window defined by by. Default is NULL
xlab	character, x-axis label. Default is "Genomic Region (5'->3')"
ylab	character, y-axis label. Default is "Peak Count Frequency"

facet	character, one of "none", "row", or "column". Controls how multiple datasets are displayed: "none" overlays them on the same plot, "row" creates row facets, "column" creates column facets. Default is "row" for plotPeakProf, "none" for others
free_y	logical, if TRUE and faceting is used, y-axis scales independently for each facet. Default is TRUE
verbose	logical, whether to print progress messages. Default is TRUE
nbin	integer, number of bins for binning mode. If provided, regions are divided into this many equal-sized bins instead of using absolute positions. Useful for comparing features of different lengths. If NULL, uses absolute positions. Default is NULL
ignore_strand	logical, whether to ignore strand information when calculating coverage. If FALSE, peaks and features are matched by strand. Default is FALSE
...	additional arguments passed to ggplot2 functions for plot customization

Details

Plot peak profiles around genomic features

This function visualizes the distribution of ChIP-seq peaks around genomic features (e.g., TSS, gene body, exons) by plotting average peak profiles or heatmaps.

The function behavior depends on the input:

- If tagMatrix is provided: plots directly from the tag matrix (supports both normal and binning modes)
- If tagMatrix is NULL: generates tag matrix from peaks and routes to:
 - plotPeakProf2() if by has length 1
 - plotPeakProf_MultiWindows() if by has length > 1

The function supports:

- Single or multiple peak sets (list of peaks)
- Confidence intervals for average profiles
- Binning mode for large datasets
- Faceting options for comparing multiple datasets

TxDb parameter can accept txdb object. But many regions can not be obtained by txdb object. In this case, Users can provide self-made granges served the same role as txdb object and pass to TxDb object.

by the features of interest.

(1) if users use txdb, by can be one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', 'UTR'. These features can be obtained by functions from txdb object.

(2) if users use self-made granges object, by can be everything. Because this by will not pass to functions to get features, which is different from the case of using txdb object. This by is only used to made labels showed in picture.

type means the property of the region. one of the "start site", "end site" and "body".

upstream and downstream parameter have different usages:

- (1) if type == 'body', upstream and downstream can use to extend the flank of body region.
- (2) if type == 'start_site'/'end_site', upstream and downstream refer to the upstream and downstream of the start_site or the end_site.

weightCol refers to column in peak file. This column acts as a weight value. Details see <https://github.com/YuLab-SMU/ChIPseeker/issues/15>

nbin refers to the number of bins. getTagMatrix() provide a binning method to get the tag matrix.

There are two ways input a list of window.

- (1) Users can input a list of self-made granges objects
- (2) Users can input a list of by and only one type. In this way, plotPeakProf_MultiWindows() can made a list of window from txdb object based on by and type.

Warning:

- (1) All of these window should be the same type. It means users can only compare a list of "start site"/"end site"/"body region" with the same upstream and downstream.
- (2) So it will be only one type and several by.
- (3) Users can make window by txdb object or self-made granges object. Users can only choose one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR' or 'UTR' in the way of using txdb object. User can input any by in the way of using self-made granges object.
- (4) Users can mingle the by designed for the two ways. plotPeakProf_MultiWindows can accpet the hybrid by. But the above rules should be followed.

<https://github.com/YuLab-SMU/ChIPseeker/issues/189>

Value

ggplot object

plotPeakProf2

plotPeakProf2

Description

plot the profile of peaks automatically

Usage

```
plotPeakProf2(
  peak,
  upstream,
  downstream,
  conf,
  by,
  type,
  weightCol = NULL,
```

```

TxDb = NULL,
xlab = "Genomic Region (5'->3')",
ylab = "Peak Count Frequency",
facet = "none",
free_y = TRUE,
verbose = TRUE,
nbin = NULL,
ignore_strand = FALSE,
...
)

```

Arguments

peak	peak file (BED format) or GRanges object, or a list of peak files/GRanges objects for comparing multiple datasets
upstream	numeric, upstream distance in base pairs. Interpretation depends on type: for "start_site"/"end_site", it's the distance upstream of the site; for "body", it's the extension beyond the feature body
downstream	numeric, downstream distance in base pairs. Interpretation depends on type: for "start_site"/"end_site", it's the distance downstream of the site; for "body", it's the extension beyond the feature body
conf	numeric, confidence level (e.g., 0.95 for 95 confidence intervals are displayed. If NA or missing, no confidence intervals are shown
by	character, feature of interest. If using TxDb: one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', 'UTR'. If using self-made GRanges: can be any label (e.g., "enhancer", "promoter") used only for plot labels
type	character, one of "start_site", "end_site", or "body". Determines which part of the feature to plot
weightCol	character, name of a metadata column in the GRanges object to use as weights for coverage calculation. If NULL, all peaks have equal weight. Default is NULL
TxDb	TxDb or EnsDb annotation object, or a GRanges object containing user-defined genomic features. Default is NULL
xlab	character, x-axis label. Default is "Genomic Region (5'->3')"
ylab	character, y-axis label. Default is "Peak Count Frequency"
facet	character, one of "none", "row", or "column". Controls how multiple datasets are displayed. Default is "none"
free_y	logical, if TRUE and faceting is used, y-axis scales independently for each facet. Default is TRUE
verbose	logical, whether to print progress messages. Default is TRUE
nbin	integer, number of bins for binning mode. If provided, regions are divided into this many equal-sized bins. If NULL, uses absolute positions. Default is NULL
ignore_strand	logical, whether to ignore strand information when calculating coverage. Default is FALSE
...	additional arguments passed to ggplot2 functions

Details

peak stands for the peak file.

by the features of interest.

(1) if users use txdb, by can be one of 'gene', 'transcript', 'exon', 'intron', '3UTR', '5UTR', 'UTR'. These features can be obtained by functions from txdb object.

(2) if users use self-made granges object, by can be everything. Because this by will not pass to functions to get features, which is different from the case of using txdb object. This by is only used to made labels showed in picture.

type means the property of the region. one of the "start site", "end site" and "body".

upstream and downstream parameter have different usages:

(1) if type == 'body', upstream and downstream can use to extend the flank of body region.

(2) if type == 'start_site'/'end_site', upstream and downstream refer to the upstream and downstream of the start_site or the end_site.

weightCol refers to column in peak file. This column acts as a weight vaule. Details see <https://github.com/YuLab-SMU/ChIPseeker/issues/15>

nbin refers to the number of bins, providing a binning method to get the tag matrix.

Txdb parameter can accept txdb object. But many regions can not be obtained by txdb object. In this case, Users can provide self-made granges served the same role as txdb object and pass to TxDb object.

plotPeakProf2() is different from the plotPeakProf(). plotPeakProf2() do not need to provide window parameter, which means plotPeakProf2() will call relevent functions to make window automatically.

Value

ggplot object

Author(s)

G Yu, Ming Li

readPeakFile

Read peak file and convert to GRanges or data.frame

Description

The function reads tab-delimited peak files and converts them to the specified output format. It automatically detects BED format files based on file extension and handles the coordinate system conversion (BED files use 0-based start coordinates, which are converted to 1-based for R compatibility). Additional columns beyond chromosome, start, and end positions are preserved in the output.

Usage

```
readPeakFile(peakfile, as = "GRanges", ...)
```

Arguments

peakfile	character, path to the peak file. Can be a local file path or a URL. Supported formats include BED (.bed, .bed.gz), narrowPeak (.narrowPeak), broadPeak (.broadPeak), gappedPeak (.gappedPeak), bedGraph (.bedGraph.gz), or any tab-delimited file with chromosome, start, and end positions
as	character, output format. One of "GRanges" (default) or "data.frame". If "GRanges", returns a GRanges object with additional columns as metadata. If "data.frame", returns a data.frame
...	additional parameters passed to <code>utils::read.delim()</code> for reading the file. Useful parameters include <code>sep</code> , <code>quote</code> , <code>stringsAsFactors</code> , etc. Note: header is automatically determined based on file format, but can be explicitly specified to override

Details

This function reads peak files in various formats (BED, narrowPeak, broadPeak, etc.) and converts them to either a GRanges object or data.frame. The function automatically handles BED file coordinate conversion (0-based to 1-based).

The function performs the following steps:

1. Detects file format based on extension (BED files: .bed, .bed.gz, .narrowPeak, .broadPeak, .gappedPeak, .bedGraph.gz, or files ending with Peak.gz)
2. Reads the file using `utils::read.delim()` with appropriate header settings (BED files have no header, other formats may have headers)
3. Converts BED coordinates from 0-based to 1-based by adding 1 to the start position (BED end position is already exclusive, so no adjustment needed)
4. Converts to the requested output format (GRanges or data.frame)
5. Preserves all additional columns beyond the first three (chromosome, start, end) as metadata columns in GRanges or as additional columns in data.frame

Supported file formats include standard BED files, narrowPeak, broadPeak, gappedPeak, bedGraph, and other tab-delimited formats with chromosome, start, and end positions in the first three columns.

Value

Depending on the `as` parameter:

- If `as="GRanges"`: A GRanges object with:
 - `seqnames`: Chromosome names (from column 1)
 - `ranges`: IRanges with start and end positions (from columns 2 and 3)
 - Additional metadata columns: All columns beyond the first three are preserved as metadata columns with their original names
- If `as="data.frame"`: A data.frame with all columns from the original file, with BED coordinates converted to 1-based if applicable

Author(s)

G Yu

Examples

```
peakfile <- system.file("extdata", "sample_peaks.txt", package="ChIPseeker")
peak.gr <- readPeakFile(peakfile, as="GRanges")
peak.gr

## Read a BED file
## peak.gr <- readPeakFile("peaks.bed", as="GRanges")

## Read with custom parameters
## peak.df <- readPeakFile("peaks.txt", as="data.frame", sep="\t")
```

rename.GRanges

*Rename columns in GRanges objects using dplyr syntax***Description**

The function renames metadata columns in GRanges objects by converting them to a data.frame, applying the dplyr rename operation, and converting back to GRanges. This allows renaming any metadata column while preserving all genomic range information.

Usage

```
## S3 method for class 'GRanges'
rename(x, ...)
```

Arguments

x	GRanges object to rename columns in
...	name-value pairs where the name is the new column name and the value (unquoted) is the old column name. Syntax: new_name = old_name. Multiple re-names can be specified in a single call

Details

This function extends `dplyr::rename()` to work with GRanges objects, allowing users to rename metadata columns using familiar dplyr syntax while preserving the GRanges structure.

The function performs the following steps:

1. Converts GRanges to data.frame using `as.data.frame()`
2. Applies `dplyr::rename()` with the provided name mappings
3. Converts back to GRanges using `makeGRangesFromDataFrame()` with `keep.extra.columns=TRUE`

All genomic range information (seqnames, ranges, strand) is preserved. Only metadata columns are renamed; standard GRanges columns (seqnames, start, end, width, strand) cannot be renamed through this function.

Value

GRanges object with renamed metadata columns. All genomic range information and other meta-data columns are preserved unchanged

Examples

```
## Not run:
## Rename a single column
peak.gr <- peak.gr %>% rename(geneSymbol = SYMBOL)

## Rename multiple columns
peak.gr <- peak.gr %>%
  rename(
    geneSymbol = SYMBOL,
    geneName = GENENAME,
    distance = distanceToTSS
  )

## Rename to remove spaces or special characters
peak.gr <- peak.gr %>% rename(gene_id = "gene Id")

## End(Not run)
```

seq2gene

Annotate genomic regions to genes in many-to-many mapping

Description

The function maps genomic regions to genes through three mechanisms:

1. **Host genes:** Regions that overlap with exons or introns of genes. These are genes that directly contain the genomic regions
2. **Proximal genes:** Regions located within the promoter region (TSS region) of genes. These are genes whose transcription start sites are near the genomic regions
3. **Flanking genes:** Regions located within a specified distance from genes but outside the promoter region. These are nearby genes that may be regulated via cis-regulatory mechanisms

The function returns a vector of unique gene IDs that can be used for downstream functional enrichment analysis (e.g., GO, KEGG, Reactome pathway analysis).

Usage

```
seq2gene(seq, tssRegion, flankDistance, TxDb, sameStrand = FALSE)
```

Arguments

<code>seq</code>	GRanges object containing genomic regions to be mapped to genes. Typically ChIP-seq peaks, enhancers, or other regulatory elements
<code>tssRegion</code>	numeric vector of length 2 specifying the promoter region around the transcription start site (TSS). Default example: <code>c(-1000, 1000)</code> means 1kb upstream and 1kb downstream of TSS. Negative values indicate upstream, positive values indicate downstream. Regions within this range are considered to be in promoter regions
<code>flankDistance</code>	numeric, distance in base pairs to search for flanking genes. Regions that are within this distance from a gene (but outside the <code>tssRegion</code>) are considered flanking genes. Default example: 3000 (3kb)
<code>TxDb</code>	TxDb or EnsDb annotation object containing gene/transcript structure information
<code>sameStrand</code>	logical, whether to only consider genes on the same strand as the genomic regions when finding overlaps and nearest features. If FALSE, strand is ignored. Default is FALSE

Details

This function associates genomic regions (e.g., ChIP-seq peaks, enhancers, regulatory elements) with coding genes in a many-to-many mapping. It is designed to link both coding and non-coding genomic regions to coding genes to facilitate functional enrichment analysis.

The function performs the following steps:

1. Identifies host genes by finding overlaps with exons and introns using `getGenomicAnnotation.internal()`
2. Extracts gene IDs from overlapping exons and introns, converting transcript-based IDs to gene IDs
3. Finds the nearest gene to each region using `getNearestFeatureIndicesAndDistances()`
4. Categorizes nearest genes into:
 - Promoter genes: genes where the distance falls within `tssRegion`
 - Flanking genes: genes where the absolute distance is less than `flankDistance` but outside `tssRegion`
5. Combines all gene IDs (host, promoter, and flanking) and returns unique gene IDs

This many-to-many mapping allows a single genomic region to be associated with multiple genes (e.g., overlapping multiple genes or having multiple nearby genes), and a single gene can be associated with multiple genomic regions.

Value

Character vector of unique gene IDs. The vector contains gene IDs from:

- Host genes (regions overlapping exons/introns)
- Proximal genes (regions within TSS region)
- Flanking genes (regions within `flankDistance` but outside TSS region)

The gene IDs are typically Entrez IDs (for TxDb) or `gene_id` (for EnsDb), and can be used directly for functional enrichment analysis with packages like `clusterProfiler`, `DOSE`, or `ReactomePA`

Author(s)

Guangchuang Yu

See Also

[annotatePeak](#) for detailed peak annotation, [getNearestFeatureIndicesAndDistances](#) for finding nearest features

Examples

```
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
TxDb <- TxDb.Hsapiens.UCSC.hg19.knownGene
file <- getSampleFiles()[[1]] # a bed file
gr <- readPeakFile(file)
genes <- seq2gene(gr, tssRegion=c(-1000, 1000), flankDistance = 3000, TxDb)

## Use the gene IDs for enrichment analysis
## library(clusterProfiler)
## ego <- enrichGO(gene = genes, OrgDb = org.Hs.eg.db, ont = "BP")

## End(Not run)
```

shuffle

Shuffle peak positions across the genome

Description

The function generates a null distribution for permutation testing by randomly repositioning peaks within their respective chromosomes. It maintains:

- The number of peaks per chromosome
- The width of each peak
- The chromosome assignment

Usage

```
shuffle(peak.gr, TxDb)
```

Arguments

peak.gr	GRanges object containing peaks to shuffle
TxDb	TxDb or EnsDb annotation object providing chromosome lengths for valid position ranges

Details

This function randomly shuffles peak positions across chromosomes while preserving the original peak widths and chromosome distribution.

The shuffling process:

1. Groups peaks by chromosome
2. For each chromosome, randomly samples new start positions from the valid range (1 to chromosome length - peak width)
3. Creates new GRanges with shuffled positions but original widths
4. Sets strand to "*" (unstranded)

This is used in permutation testing to generate null distributions for statistical significance testing.

Value

GRanges object with shuffled peak positions. Peaks are randomly repositioned within their chromosomes, maintaining original widths and chromosome distribution

Author(s)

G Yu

subset.csAnno	<i>Subset csAnno object</i>
---------------	-----------------------------

Description

The function allows subsetting of csAnno objects using the same syntax as subsetting GRanges objects. It filters the peaks in the anno slot and automatically updates:

- The detailGenomicAnnotation data frame to match the filtered peaks
- The annoStat annotation statistics
- The peakNum total peak count

Usage

```
## S3 method for class 'csAnno'
subset(x, ...)
```

Arguments

x	csAnno object to subset
...	logical expressions indicating which peaks to keep. Can use any column names from the GRanges metadata (e.g., annotation, distanceToTSS, geneId, etc.). See subset for details

Details

This function subsets a `csAnno` object by filtering peaks based on logical conditions, and automatically updates the associated annotation statistics and detail annotations.

The function:

1. Creates an index based on chromosome, start, and end positions of all peaks
2. Subsets the `GRanges` object in the `anno` slot using the provided conditions
3. Filters the `detailGenomicAnnotation` data frame to keep only rows corresponding to the remaining peaks
4. Recalculates annotation statistics using `getGenomicAnnoStat()`
5. Updates the peak count

The `tssRegion`, `level`, and `hasGenomicAnnotation` slots remain unchanged.

Value

A `csAnno` object containing only the filtered peaks, with updated annotation statistics and detail annotations

Author(s)

G Yu

Examples

```
## Not run:
## Subset peaks by annotation type
peakAnno_promoter <- subset(peakAnno, annotation == "Promoter")

## Subset peaks by distance to TSS
peakAnno_near <- subset(peakAnno, abs(distanceToTSS) < 5000)

## Subset by multiple conditions
peakAnno_filtered <- subset(peakAnno,
                           annotation == "Promoter" &
                           abs(distanceToTSS) < 3000)

## End(Not run)
```

tagHeatmap

tagHeatmap

Description

plot the heatmap of tagMatrix

Usage

```
tagHeatmap(
  tagMatrix,
  xlab = "",
  ylab = "",
  title = NULL,
  palette = "RdBu",
  nrow = NULL,
  ncol = NULL
)
```

Arguments

tagMatrix	matrix or list of matrices. Tag matrix with features as rows and genomic positions as columns, typically obtained from <code>getTagMatrix()</code> . If a list is provided, creates separate heatmaps for each matrix
xlab	character, x-axis label. Default is "" (empty)
ylab	character, y-axis label. Default is "" (empty)
title	character, plot title. If NULL, no title is shown. Default is NULL
palette	character, color palette name from RColorBrewer for the heatmap. See scale_colour_brewer for options. Default is "RdBu"
nrow	integer, number of rows for arranging multiple heatmaps when tagMatrix is a list. If NULL, automatically determined. Default is NULL
ncol	integer, number of columns for arranging multiple heatmaps when tagMatrix is a list. If NULL, automatically determined. Default is NULL

Value

figure

Author(s)

G Yu

upsetplot.csAnno

Create UpSet plot for genomic annotation combinations

Description

The function creates an UpSet plot (an alternative to Venn diagrams) that shows how peaks are distributed across different combinations of genomic annotations. It displays:

- The frequency of each annotation combination (intersection)
- Which annotations are present in each combination
- Optionally, a vennpie plot overlay showing the overall annotation distribution

Usage

```
upsetplot.csAnno(
  x,
  order_by = "freq",
  vennpie = FALSE,
  vp = list(x = 0.6, y = 0.7, width = 0.8, height = 0.8)
)
```

Arguments

x	csAnno object containing annotated peaks with detailGenomicAnnotation slot
order_by	character, how to order the intersections in the plot. Options: "freq" (by frequency, default) or "degree" (by number of sets in the intersection). Default is "freq"
vennpie	logical, whether to overlay a vennpie plot showing the overall annotation distribution. If TRUE, the vennpie is embedded as a subplot. Default is FALSE
vp	list, viewport parameters for positioning the vennpie subplot when vennpie=TRUE. Contains: • x: x position (0-1, left to right). Default is 0.6 • y: y position (0-1, bottom to top). Default is 0.7 • width: width of subplot (0-1). Default is 0.8 • height: height of subplot (0-1). Default is 0.8

Details

This function visualizes the combinations of genomic annotations (e.g., Promoter, Exon, Intron, UTR) that peaks can have, using an UpSet plot to show set intersections and their frequencies.

The function performs the following steps:

1. Extracts the detailGenomicAnnotation data frame from the csAnno object, which contains logical columns indicating which annotations each peak has
2. Converts the annotation matrix into a list format where each peak is represented by the set of annotations it has
3. Creates an UpSet plot using ggupset showing:
 - Left panel: bar chart of intersection sizes (frequencies)
 - Top panel: bar chart showing individual set sizes
 - Bottom panel: matrix showing which sets are included in each intersection
4. Optionally overlays a vennpie plot if vennpie=TRUE

UpSet plots are particularly useful when there are many annotation categories and peaks can have multiple annotations simultaneously (e.g., a peak can be both in a Promoter and overlap an Exon).

Value

A ggplot2 object (or combined plot if vennpie=TRUE) showing the UpSet visualization. The plot can be further customized using ggplot2 functions or printed directly

Author(s)

Guangchuang Yu

See Also[vennpie](#) for the vennpie plot function, [plotAnnoBar](#) for bar plots of annotation distribution**Examples**

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfile <- system.file("extdata", "sample_peaks.txt", package="ChIPseeker")
peakAnno <- annotatePeak(peakfile, TxDb=txdb)

## Basic UpSet plot
upsetplot(peakAnno)

## UpSet plot with vennpie overlay
upsetplot(peakAnno, vennpie=TRUE)

## Order by degree instead of frequency
upsetplot(peakAnno, order_by="degree")

## End(Not run)
```

vennpie*Create Venn pie chart for genomic annotation distribution*

Description

The function visualizes the genomic annotation of peaks using a nested pie chart approach. It displays three concentric pie charts showing:

- **Outer ring:** Genic vs Intergenic distribution
- **Middle ring:** Breakdown of Genic (Intron, Exon) and Intergenic (Upstream, Downstream, Distal Intergenic) categories
- **Inner ring:** Further breakdown showing Exon and Downstream regions in detail

This visualization helps understand the overlap between different annotation categories, as peaks can be annotated with multiple features simultaneously.

Usage

```
vennpie(x, r = 0.2, cex = 1.2, ...)
```

Arguments

<code>x</code>	csAnno object containing annotated peaks with detailGenomicAnnotation slot populated
<code>r</code>	numeric, initial radius for the base pie chart. Controls the overall size of the plot. The nested pies use multiples of this radius (2*r, 3*r, 4*r). Default is 0.2
<code>cex</code>	numeric, character expansion factor for the legend text. Larger values make the legend text bigger. Default is 1.2
<code>...</code>	additional parameters passed to the plotting functions

Details

This generic function creates a nested pie chart (venn diagram style) visualization showing the distribution of peaks across different genomic annotation categories, with hierarchical nesting to show overlapping annotations.

Value

A plot showing nested pie charts with genomic annotation distribution

See Also

[plotAnnoBar](#) [plotAnnoPie](#) for alternative visualization methods

vennpie.csAnno

Create Venn pie chart for genomic annotation distribution

Description

The function visualizes the genomic annotation of peaks using a nested pie chart approach. It displays three concentric pie charts showing:

- **Outer ring:** Genic vs Intergenic distribution
- **Middle ring:** Breakdown of Genic (Intron, Exon) and Intergenic (Upstream, Downstream, Distal Intergenic) categories
- **Inner ring:** Further breakdown showing Exon and Downstream regions in detail

This visualization helps understand the overlap between different annotation categories, as peaks can be annotated with multiple features simultaneously (e.g., a peak can be both in an Exon and in a Promoter region).

Usage

```
vennpie.csAnno(x, r = 0.2, cex = 1.2, col = NULL)
```

Arguments

x	csAnno object containing annotated peaks with detailGenomicAnnotation slot populated
r	numeric, initial radius for the base pie chart. Controls the overall size of the plot. The nested pies use multiples of this radius (2*r, 3*r, 4*r). Default is 0.2
cex	numeric, character expansion factor for the legend text. Larger values make the legend text bigger. Default is 1.2
col	named character vector, custom colors for annotation categories. Names should match category names: "Genic", "Intergenic", "Intron", "Exon", "Upstream", "Downstream", "Distal_Intergenic". Colors not specified will use default colors. Default is NULL (uses default color scheme)

Details

This function creates a nested pie chart (venn diagram style) visualization showing the distribution of peaks across different genomic annotation categories, with hierarchical nesting to show overlapping annotations.

The function performs the following steps:

1. Extracts detailed genomic annotation from the csAnno object
2. Calculates counts for each annotation category:
 - Genic: peaks overlapping exons or introns
 - Intergenic: peaks not in genic regions
 - Exon: peaks overlapping exons
 - Intron: peaks overlapping introns
 - Upstream: peaks in promoter regions with negative distance to TSS
 - Downstream: peaks downstream of genes
 - Distal Intergenic: intergenic peaks far from genes
3. Creates three nested floating pie charts using `floating.pie()` from the `plotrix` package
4. Adds a legend showing all annotation categories with their colors

The function uses pseudo-counts (+1) for each category to ensure proper visualization even when some categories have zero counts, preventing color mismatches in the pie charts.

Value

No return value. Creates a plot showing nested pie charts of genomic annotation distribution

Author(s)

G Yu

See Also

[plotAnnoPie](#) for a simple pie chart, [plotAnnoBar](#) for a bar chart visualization

Examples

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfile <- system.file("extdata", "sample_peaks.txt", package="ChIPseeker")
peakAnno <- annotatePeak(peakfile, TxDb=txdb)

## Create venn pie chart
vennpie(peakAnno)

## Customize colors
vennpie(peakAnno, col=c(Exon="red", Intron="blue"))

## Adjust size and legend
vennpie(peakAnno, r=0.3, cex=1.5)

## End(Not run)
```

vennplot

Plot Venn diagram for overlapping sets

Description

The function calculates all possible intersections between sets using the `overlap()` function, then visualizes the results as a Venn diagram. It supports three plotting backends, each with different features:

- `gplots`: Default method, produces simple black-and-white Venn diagrams using the `gplots` package
- `ggVennDiagram`: Produces colorful, customizable Venn diagrams using `ggplot2`, allowing full control over colors, labels, and styling
- `Vennerable`: Alternative method using the `Vennerable` package (requires installation from R-Forge)

Usage

```
vennplot(Sets, by = "gplots", ...)
```

Arguments

Sets named list of objects to compare. Each element can be:

- A vector (e.g., gene IDs, peak names)
- A `GRanges` object (genomic regions)

If the list is not named, names will be auto-generated as "Set1", "Set2", etc.

by character, plotting method to use. One of:

- "gplots" (default): Simple black-and-white Venn diagrams

- "ggVennDiagram": Colorful, customizable ggplot2-based diagrams
- "Vennerable": Alternative method (requires Vennerable package)

... additional parameters passed to `ggVennDiagram::ggVennDiagram()` when `by="ggVennDiagram"`.
Common parameters include:

- `show_percentage`: logical, whether to show percentages
- `label_alpha`: numeric, transparency of labels
- `edge_size`: numeric, size of circle edges

See [ggVennDiagram](#) for all available options

Details

This function creates Venn diagrams to visualize overlaps between multiple sets of objects (e.g., peaks, genes, genomic regions). It supports three different plotting methods with different features and customization options.

The function performs the following steps:

1. Validates input: ensures the list is named (auto-generates names if missing, e.g., "Set1", "Set2", ...)
2. Calculates all possible intersections using `overlap()`, which handles both vectors and `GRanges` objects
3. Creates the Venn diagram using the specified method:
 - `gplots`: Uses `plot.venn()` with a matrix format
 - `ggVennDiagram`: Passes sets directly to `ggVennDiagram::ggVennDiagram()` with additional parameters
 - `Vennerable`: Creates a Venn object and plots it

For `GRanges` objects, the function uses set intersection to find overlapping genomic regions. For vectors, it uses standard set intersection.

Value

A Venn diagram plot. The return type depends on the `by` parameter:

- `gplots`: Base R plot (invisible return)
- `ggVennDiagram`: ggplot2 object that can be further customized
- `Vennerable`: Grid plot (invisible return)

Author(s)

G Yu

See Also

[overlap](#) for calculating set intersections, [vennplot.peakfile](#) for a convenience function for peak files

Examples

```
## Not run:
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakfiles <- getSampleFiles()
peakAnnoList <- lapply(peakfiles, annotatePeak, TxDb=txdb)
names(peakAnnoList) <- names(peakfiles)

## Compare genes from different peak files
genes <- lapply(peakAnnoList, function(i) as.data.frame(i)$geneId)
vennplot(genes)

## Use ggVennDiagram for customizable plot
vennplot(genes, by="ggVennDiagram", show_percentage=TRUE)

## Compare GRanges objects directly
peaks <- lapply(peakfiles, readPeakFile)
names(peaks) <- names(peakfiles)
vennplot(peaks)

## End(Not run)
```

vennplot.peakfile	<i>Plot Venn diagram for peak files</i>
-------------------	---

Description

The function simplifies the workflow of comparing multiple peak files by automatically reading the files and setting up the Venn diagram. It loads each peak file using `readPeakFile()`, optionally labels them, and then calls `vennplot()` to create the visualization.

Usage

```
vennplot.peakfile(files, labels = NULL)
```

Arguments

<code>files</code>	character vector of peak file paths. Supported formats include BED, narrow-Peak, broadPeak, and other formats supported by <code>readPeakFile()</code>
<code>labels</code>	character vector of labels for each peak file. If <code>NULL</code> , labels are automatically generated from filenames by removing file extensions. Default is <code>NULL</code>

Details

This is a convenience function that reads peak files and creates a Venn diagram showing the overlap of genomic regions between different peak sets.

The function:

1. Reads each peak file using `readPeakFile()` (converts to `GRanges`)
2. Generates labels from filenames if not provided (removes file extensions)
3. Names the peak list with the labels
4. Calls `vennplot()` to create the Venn diagram

This is particularly useful for quickly comparing multiple ChIP-seq peak files to see how many peaks are shared between experiments.

Value

A Venn diagram plot showing the overlap of peaks between different files. The plot is created using the default method (by="gplots"). See [vennplot](#) for details on the return type

Author(s)

G Yu

See Also

[vennplot](#) for the main plotting function, [readPeakFile](#) for reading peak files

Examples

```
## Not run:
## Compare multiple peak files
peakfiles <- c("sample1.bed", "sample2.bed", "sample3.bed")
vennplot.peakfile(peakfiles)

## With custom labels
vennplot.peakfile(peakfiles, labels=c("Treatment", "Control", "Input"))

## Using getSampleFiles()
files <- getSampleFiles()
vennplot.peakfile(files)

## End(Not run)
```

Index

- * **classes**
 - csAnno-class, [17](#)
- * **datasets**
 - info, [37](#)
- * **package**
 - ChIPseeker-package, [3](#)
- .., [5](#)
- .ChIPseekerEnv, [6](#)
- annotatePeak, [3](#), [4](#), [7](#), [15](#), [17](#), [49](#), [51](#), [52](#), [56](#), [58](#), [68](#)
- arrange.GRanges, [10](#)
- as.data.frame.csAnno, [12](#)
- as.GRanges, [12](#), [13](#)
- ChIPseeker (ChIPseeker-package), [3](#)
- ChIPseeker-package, [3](#)
- combine_csAnno, [14](#)
- covplot, [4](#), [15](#)
- csAnno-class, [17](#)
- downloadGEObedFiles, [4](#), [18](#)
- downloadGSMbedFiles, [4](#), [18](#)
- dropAnno, [19](#)
- enrichAnnoOverlap, [4](#), [19](#)
- enrichPeakOverlap, [4](#), [21](#)
- filter.GRanges, [23](#)
- getAnnoStat, [25](#)
- getBioRegion, [4](#), [26](#), [33](#), [36](#), [39](#)
- getGeneAnno, [27](#)
- getGEOgenomeVersion, [4](#), [29](#)
- getGEOinfo, [4](#), [29](#)
- getGEOspecies, [4](#), [30](#)
- getNearestFeatureIndicesAndDistances, [30](#), [68](#)
- getPromoters, [4](#), [27](#), [32](#), [36](#)
- getSampleFiles, [33](#)
- getTagMatrix, [4](#), [34](#)
- getTagMatrix.binning.internal, [36](#)
- ggVennDiagram, [77](#)
- gsminfo (info), [37](#)
- info, [37](#)
- makeBioRegionFromGranges, [27](#), [36](#), [38](#)
- mclapply, [22](#)
- mutate.GRanges, [39](#)
- overlap, [41](#), [77](#)
- p.adjust, [20](#), [22](#)
- peak_Profile_Heatmap, [42](#)
- peakHeatmap, [4](#), [44](#)
- peakHeatmap_multiple_Sets, [46](#)
- plotAnnoBar, [4](#), [10](#), [25](#), [48](#), [51](#), [52](#), [56](#), [73–75](#)
- plotAnnoBar, csAnno-method (csAnno-class), [17](#)
- plotAnnoBar, list-method (plotAnnoBar), [48](#)
- plotAnnoBar.data.frame, [49](#)
- plotAnnoPie, [4](#), [10](#), [49](#), [50](#), [74](#), [75](#)
- plotAnnoPie, csAnno-method (csAnno-class), [17](#)
- plotAnnoPie.csAnno, [51](#)
- plotAvgProf, [4](#), [36](#), [52](#)
- plotAvgProf2, [54](#)
- plotDistToTSS, [4](#), [10](#), [55](#)
- plotDistToTSS, csAnno-method (csAnno-class), [17](#)
- plotDistToTSS, list-method (plotDistToTSS), [55](#)
- plotDistToTSS.data.frame, [56](#)
- plotPeakProf, [4](#), [36](#), [58](#)
- plotPeakProf2, [61](#)
- readPeakFile, [4](#), [63](#), [79](#)
- rename.GRanges, [65](#)
- scale_colour_brewer, [43](#), [45](#), [71](#)

seq2gene, [3](#), [4](#), [66](#)
show, csAnno-method (csAnno-class), [17](#)
shuffle, [4](#), [68](#)
subset, [69](#)
subset, csAnno-method (csAnno-class), [17](#)
subset.csAnno, [69](#)

tagHeatmap, [4](#), [70](#)
tagMatrixList (info), [37](#)

ucsc_release (info), [37](#)
upsetplot, [4](#)
upsetplot, csAnno-method (csAnno-class),
[17](#)
upsetplot.csAnno, [71](#)

vennpie, [4](#), [51](#), [73](#), [73](#)
vennpie, csAnno-method (csAnno-class), [17](#)
vennpie.csAnno, [74](#)
vennplot, [4](#), [76](#), [79](#)
vennplot.peakfile, [77](#), [78](#)