# Package 'BindingSiteFinder'

April 23, 2025

**Type** Package

**Title** Binding site defintion based on iCLIP data

**Version** 2.6.0

**Description** Precise knowledge on the binding sites of an RNA-binding protein (RBP) is key to understand (post-) transcriptional regulatory processes. Here we present a workflow that describes how exact binding sites can be defined from iCLIP data. The package provides functions for binding site definition and result visualization. For details please see the vignette.

**License** Artistic-2.0

**Encoding** UTF-8

**VignetteBuilder** knitr

**Imports** tidyr, tibble, plyr, matrixStats, stats, ggplot2, methods, rtracklayer, S4Vectors, ggforce, GenomeInfoDb, ComplexHeatmap, RColorBrewer, lifecycle, rlang, forcats, dplyr, GenomicFeatures, IRanges, kableExtra, ggdist

**Depends** GenomicRanges, R (>= 4.2)

**Suggests** testthat, BiocStyle, knitr, rmarkdown, GenomicAlignments, scales, Gviz, xlsx, GGally, patchwork, viridis, ggplotify, SummarizedExperiment, DESeq2, ggpointdensity, ggrastr, ashr, txdbmaker, ggrepel, stringr

**RoxygenNote** 7.3.2

**Collate** 'AllClasses.R' 'AllGenerics.R' 'Functions.R' 'methods.R' 'bindingsites.R' 'helper.R' 'PlotFunction.R' 'CoverageFunctions.R' 'workflow.R' 'helperSpecific.R' 'exports.R' 'coveragePlots.R' 'bsfind.R' 'helperPlots.R' 'differentialFunctions.R' 'differentialPlots.R'

**biocViews** Sequencing, GeneExpression, GeneRegulation, FunctionalGenomics, Coverage, DataImport

**BugReports** https://github.com/ZarnackGroup/BindingSiteFinder/issues

**git_url** https://git.bioconductor.org/packages/BindingSiteFinder

**git_branch** RELEASE_3_21

**git_last_commit** efff5d6

**git_last_commit_date** 2025-04-15

**Repository** Bioconductor 3.21

**Date/Publication** 2025-04-23

**Author** Mirko Brüggemann [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1778-0248>),
Melina Klostermann [aut] (ORCID:
<https://orcid.org/0000-0003-3122-1095>),
Kathi Zarnack [aut] (ORCID: <https://orcid.org/0000-0003-3527-3378>)

**Maintainer** Mirko Brüggemann <mirko.brueggemann@mail.de>

# Contents

---

`add-BSFDataSet` *Add two* `BSFDataSet` *objects*

---

## Description

Use '+' to add two objects of type `BSFDataSet` to each other.

## Usage

```
## S4 method for signature 'BSFDataSet,BSFDataSet'
e1 + e2
```

## Arguments

| | |
|---|---|
| e1 | BSFDataSet; the first dataset |
| e2 | BSFDataSet; the second dataset |

## Details

Meta data is extended by binding both input tables together. Ranges are added by re-centering partially overlapping ranges according to their combined coverage maximum.

Input ranges must be of the same size. Differently size objects cannot be added. For this and other usecases please see combineBSF.

## Value

A BSFDataSet object with ranges, signal and meta data from both inputs.

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# make binding sites
bds = makeBindingSites(bds, bsSize = 7)

# split ranges in two groups
allRanges = getRanges(bds)
set.seed(1234)
idx = sample(1:length(allRanges), size = length(allRanges)/2, replace = FALSE)
r1 = allRanges[idx]
r2 = allRanges[-idx]

# splite meta data
allMeta = getMeta(bds)
m1 = allMeta[1:2,]
m2 = allMeta[3:4,]

# create new objects
bds1 = setRanges(bds, r1)
bds2 = setRanges(bds, r2)
bds1 = setMeta(bds1, m1)
bds2 = setMeta(bds2, m2)
bds1 = setName(bds1, "test1")
bds2 = setName(bds2, "test2")

# merge two objects with '+' operator
c1 = bds1 + bds2
```

annotateWithScore *Annotation function for BSFDataSet object*

## Description

This function can be used to annotate a BSFDataSet object with merged binding sites with scores from the initial ranges (eg. PureCLIP scores).

## Usage

```
annotateWithScore(
  object,
  match.ranges = NULL,
  match.score = "score",
  match.option = c("max", "sum", "mean"),
  scoreRanges = lifecycle::deprecated(),
  MatchColScore = lifecycle::deprecated(),
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| match.ranges | a GRanges object, with numeric column for the score to match |
| match.score | character; meta column name of the crosslink site GenomicRanges object that holds the score to match |
| match.option | character; option how score should be matched |
| scoreRanges | deprecated -> use match.ranges instead |
| MatchColScore | deprecated -> use match.score instead |
| quiet | logical; whether to print messages |

## Details

The function is part of the standard workflow performed by BSFind.

## Value

an object of class BSFDataSet with updated meta columns of the ranges

## See Also

BSFind, globalScorePlot

## Examples

```
if (.Platform$OS.type != "windows") {
    # load data
    csFile <- system.file("extdata", "PureCLIP_crosslink_sites_examples.bed",
                        package="BindingSiteFinder")
    cs = rtracklayer::import(con = csFile, format = "BED",
    extraCols=c("additionalScores" = "character"))
    cs$additionalScores = NULL
    clipFiles <- system.file("extdata", package="BindingSiteFinder")
    # two experimental conditions
    meta = data.frame(
    id = c(1,2,3,4),
    condition = factor(c("WT", "WT", "KD", "KD"),
    levels = c("KD", "WT")),
    clPlus = list.files(clipFiles, pattern = "plus.bw$", full.names = TRUE),
    clMinus = list.files(clipFiles, pattern = "minus.bw$",
     full.names = TRUE))
    bds = BSFDataSetFromBigWig(ranges = cs, meta = meta, silent = TRUE)

    # merge binding sites
    bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
    minCrosslinks = 2, minClSites = 1)

    # annotate with original pureCLIP score
    bdsRe = annotateWithScore(bds, cs)
}
```

---

`assignToGenes` *Assign binding sites to their hosting genes*

---

## Description

Function that assigns each binding site in the `BSFDataSet` to its hosting gene given a gene annotation (`anno.annoDB`, `anno.genes`).

## Usage

```
assignToGenes(
  object,
  overlaps = c("frequency", "hierarchy", "remove", "keep"),
  overlaps.rule = NULL,
  anno.annoDB = NULL,
  anno.genes = NULL,
  match.geneID = "gene_id",
  match.geneName = "gene_name",
  match.geneType = "gene_type",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored binding sites. This means that ranges should be > 1 |
| overlaps | character; how overlapping gene loci should be handled. |
| overlaps.rule | character vector; a vector of gene type that should be used to handle overlapping cases in a hierarchical manor. The order of the vector is the order of the hierarchy. |
| anno.annoDB | an object of class OrganismDbi that contains the gene annotation (!!! Experimental !!!). |
| anno.genes | an object of class GenomicRanges that represents the gene ranges directly |
| match.geneID | character; meta column name of the gene ID |
| match.geneName | character; meta column name of the gene name |
| match.geneType | character; meta column name of the gene type |
| quiet | logical; whether to print messages |

## Details

Regardless of the annotation source that is being used, the respective meta information about the genes have to be present. They can be set by the match.geneID, match.geneName and match.geneType arguments.

In the case of overlapping gene annotation, a single binding site will be associated with multiple genes. The overlaps parameter allows to decide in these cases. Option 'frequency' will take the most frequently observed gene type, option 'hierarchy' works in conjunction with a user defined rule (overlaps.rule). Options 'remove' and 'keep' will remove or keep all overlapping cases, respectively.

Note that if an overlaps exists, but gene types are identical options 'frequency' and 'hierarchy' will cause the gene that was seen first to be selected as representative.

The function is part of the standard workflow performed by BSFind.

## Value

an object of class BSFDataSet with binding sites having hosting gene information added to their meta columns.

## See Also

BSFind, geneOverlapsPlot, targetGeneSpectrumPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# Load GRanges with genes
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
bds = makeBindingSites(object = bds, bsSize = 9)
```

```
bds = assignToGenes(bds, anno.genes = gns)
```

---

`assignToTranscriptRegions` *Assign binding sites to their hosting transcript regions*

---

## Description

Function that assigns each binding site in the `BSFDataSet` to its hosting transcript region given an annotation database (`anno.annoDB`), or a GRanges list / `CompressedGRangesList` (`anno.transcriptRegionList`) that holds the ranges for the transcript regions of interest.

## Usage

```
assignToTranscriptRegions(
  object,
  overlaps = c("frequency", "hierarchy", "flag", "remove"),
  overlaps.rule = NULL,
  anno.annoDB = NULL,
  anno.transcriptRegionList = NULL,
  normalize.exclude.upper = 0.02,
  normalize.exclude.lower = 0.02,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object with stored binding sites. This means that ranges should be > 1 |
| `overlaps` | character; how overlapping transcript regions should be handled. |
| `overlaps.rule` | character vector; a vector of transcript region names that should be used to handle overlapping cases in a hierarchical manor. The order of the vector is the order of the hierarchy. |
| `anno.annoDB` | an object of class `OrganismDbi` that contains the transcript region annotation (!!! Experimental !!!). |
| `anno.transcriptRegionList` | an object of class `CompressedGRangesList` that holds an ranges for each transcript region |
| `normalize.exclude.upper` | numeric; percentage value that indicates the upper boundary for transcript region length to be considered when calculating normalization factors for regions. |
| `normalize.exclude.lower` | numeric; percentage value that indicates the lower boundary for transcript region length to be considered when calculating normalization factors for regions. |
| `quiet` | logical; whether to print messages |

## Details

Since the assignment is based on the overlaps of annotated transcript ranges and binding sites, no matching meta data is needed.

In the case of transcript regions overlaps are very frequent. To resolve these cases the `overlaps` argument can be used. Option 'frequency' will take the most frequently observed transcript region, option 'hierarchy' works in conjunction with a user defined rule (`overlaps.rule`). Options 'flag' and 'remove' will label binding sites with an ambiguous tag or remove all overlapping cases, respectively.

The function is part of the standard workflow performed by `BSFind`.

## Value

an object of class `BSFDataSet` with binding sites having hosting transcript region information added to their meta columns.

## See Also

`BSFind`, `transcriptRegionOverlapsPlot`, `transcriptRegionSpectrumPlot`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])

bds = makeBindingSites(object = bds, bsSize = 9)
bds = assignToGenes(bds, anno.genes = gns)
bds = assignToTranscriptRegions(object = bds, anno.transcriptRegionList = regions)
```

---

`bindingSiteCoveragePlot`

*Plot signal coverage of selected ranges*

---

## Description

Function plots the coverage of the CLIP data in the signal slot and plots it as coverage. The plot is centered around a given binding site, which can be selected by an index.

## Usage

```
bindingSiteCoveragePlot(
  object,
  plotIdx,
  flankPos,
```

```
  shiftPos = NULL,
  mergeReplicates = FALSE,
  autoscale = FALSE,
  highlight = TRUE,
  showCentralRange = TRUE,
  customRange = NULL,
  customRange.name = "custom",
  customAnnotation = NULL,
  customAnnotation.name = "anno",
  title = NULL,
  colorPalette = NULL
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| plotIdx | integer, index of the range to plot |
| flankPos | numeric, number of nucleotides by which the plotting frame is symmetrically extended |
| shiftPos | numeric, nucleotide positions by which the current plotting range should be shifted |
| mergeReplicates | logical, if replicates should be merge per condition (TRUE) or if every replicates should be shown separately (FALSE) |
| autoscale | logical, if y-axis should be scaled to the maximum for all replicates (TRUE), or not (FALSE) |
| highlight | logical, if the central range should be highlighted (TRUE), or not (FALSE) |
| showCentralRange | logical, if the central range should be shown (TRUE), or not (FALSE) |
| customRange | GenomicRanges, a custom range object to be shown underneath the coverage tracks |
| customRange.name | character, the name of the customRange track |
| customAnnotation | GenomicRanges or TxDb, a custom annotation for eg. gene, exons, etc. to be shown underneath the coverage tracks |
| customAnnotation.name | character, the name of the customAnnotation track |
| title | character, set plotting title |
| colorPalette | vector, hex colors used for the conditions |

## Value

an object of class GVIZ

## See Also

BSFDataSet, BSFind

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bindingSiteCoveragePlot(bds, plotIdx = 3, flankPos = 10)
```

---

# bindingSiteDefinednessPlot

*Binding site definedness plot*

---

## Description

Binding site definedness is given by the percent of crosslinks that fall diretly inside the binding site compare to those around the binding site. This plotting function shows the distribution of those percentage values grouped by what is indicated in the by argument.

## Usage

```
bindingSiteDefinednessPlot(
  object,
  by = c("all", "transcript_region", "gene_type"),
  showN.genes = 5,
  show.others = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| by | character; the option by which the plot should be grouped by. Options are: "all", "transcript_region", "gene_type" |
| showN.genes | numeric; if by is 'gene_type', then this argument set the maximum number of groups to be shown in the plot |
| show.others | logical; whether to show 'others' category. |

## Details

If by = 'all', then all binding site are grouped into one distribution. For options 'transcript_region' and 'gene_type' binding sites are split into groups according to the respective assignment. This requires that the respective assignment function was executed on the dataset prior to calling this plot function.

## Value

a plot of type ggplot

## See Also

BSFind, calculateSignalToFlankScore

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
bds = BSFind(bds, anno.genes = gns, anno.transcriptRegionList = regions,
 est.subsetChromosome = "chr22")
bds = calculateSignalToFlankScore(bds)
bindingSiteDefinednessPlot(bds)
```

---

# BSFDataSet *BSFDataSet object and constructors*

---

## Description

BSFDataSet contains the class GenomicRanges, which is used to store input ranges. Alongside with the iCLIP signal in list structure and additional meta data as data.frame.

## Usage

```
BSFDataSet(ranges, meta, signal, dropSeqlevels = TRUE, silent = FALSE)

BSFDataSet(ranges, meta, signal, dropSeqlevels = TRUE, silent = FALSE)

BSFDataSetFromBigWig(ranges, meta, silent = FALSE, dropSeqlevels = TRUE)
```

## Arguments

| | |
|---|---|
| ranges | a GenomicRanges with the desired ranges to process. The strand slot must be either + or -. |
| meta | a data.frame with at least two columns. The first column should be a unique numeric id. The second column holds sample type information, such as the condition. |
| signal | a list with the two entries 'signalPlus' and 'signalMinus', following a special representation of SimpleRleList for counts per replicates (see details for more information). |
| dropSeqlevels | enforce seqnames to be the same in ranges and signal, by dropping unused seqlevels which is required for most downstream functions such as coverageOverRanges |
| silent | suppress messages but not warnings (TRUE/ FALSE) |

## Details

The ranges are enforced to have to have a "+" or "-" strand annotation,"*" is not allowed. They are expected to be of the same width and a warning is thrown otherwise.

The meta information is stored as `data.frame` with at least two required columns, 'id' and 'condition'. They are used to build the unique identifier for each replicate split by '_' (eg. id = 1 and condition = WT will result in 1_WT).

The meta data needs to have the additional columns 'clPlus' and 'clMinus' to be present if `BSFDataSetFromBigWig` is called. It is used to provide the location to the iCLIP coverage files to the import function. On object initialization these files are loaded and internally represented in the signal slot of the object (see `BSFDataSet`).

The iCLIP signal is stored in a special list structure. At the lowest level crosslink counts per nucleotide are stored as `Rle` per chromosome summarized as a `SimpleRleList`. Such a list exits for each replicate and must be named by the replicate identifier (eg. 1_WT). Therefore this list contains always exactly the same number of entries as the number of replicates in the dataset. Since we handle strands initially seperated from each other this list must be given twice, once for each strand. The strand specific entries must be named 'signalPlus' and 'signalMinus'.

The option `dropSeqlevels` forces the seqnames of the ranges and the signal to be the same. If for a specific chromosome in the ranges no respective entry in the signal list can be found, then entries with that chromosome are dropped This behavior is needed to keep the BSFDataSet object in sync, which is required for downstream functions such as `coverageOverRanges`

## Value

A BSFDataSet object.

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
rng = getRanges(bds)
sgn = getSignal(bds)
mta = getMeta(bds)
bdsNew = BSFDataSet(ranges = rng, signal = sgn, meta = mta)
```

---

`BSFind` *RBP binding site definition for iCLIP data*

---

## Description

This is the main function that performs the binding site definition analysis through the following steps:

1. Filter PureCLIP sites by their score distribution: `pureClipGlobalFilter`

2. Estimate the appropriate binding site width together with the optimal gene-wise filter level: estimateBsWidth
3. Filter PureCLIP sites by their score distribution per gene: pureClipGeneWiseFilter
4. Define equally sized binding sites: makeBindingSites
5. Perform replicate reproducibility filter: reproducibilityFilter
6. Assign binding sites to their hosting genes: assignToGenes
7. Assign binding sites to their hosting transcript regions: assignToTranscriptRegions
8. Re-assign PureCLIP scores to binding sites: annotateWithScore
9. Calculation of signal-to-flank ratio: calculateSignalToFlankScore

## Usage

```
BSFind(
  object,
  bsSize = NULL,
  cutoff.geneWiseFilter = NULL,
  cutoff.globalFilter = 0.01,
  est.bsResolution = "medium",
  est.geneResolution = "medium",
  est.maxBsWidth = 13,
  est.minimumStepGain = 0.02,
  est.maxSites = Inf,
  est.subsetChromosome = "chr1",
  est.minWidth = 2,
  est.offset = 1,
  est.sensitive = FALSE,
  est.sensitive.size = 5,
  est.sensitive.minWidth = 2,
  merge.minWidth = 2,
  merge.minCrosslinks = 2,
  merge.minClSites = 1,
  merge.CenterIsClSite = TRUE,
  merge.CenterIsSummit = TRUE,
  repro.cutoff = NULL,
  repro.nReps = NULL,
  repro.minCrosslinks = 1,
  overlaps.geneWiseFilter = "keepSingle",
  overlaps.geneAssignment = "frequency",
  overlaps.rule.geneAssignment = NULL,
  overlaps.TranscriptRegions = "frequency",
  overlaps.rule.TranscriptRegions = NULL,
  stf.flank = "bs",
  stf.flank.size = NULL,
  match.score = "score",
  match.geneID = "gene_id",
  match.geneName = "gene_name",
  match.geneType = "gene_type",
```

```
  match.ranges.score = NULL,
  match.option.score = "max",
  anno.annoDB = NULL,
  anno.genes = NULL,
  anno.transcriptRegionList = NULL,
  quiet = FALSE,
  veryQuiet = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored ranges |
| bsSize | an odd integer value specifying the size of the output binding sites |
| cutoff.geneWiseFilter | numeric; defines the cutoff for which sites to remove in in function pureClipGeneWiseFilter. The smallest step is 1% (0.01). A cutoff of 5% will remove the lowest 5% sites, given their score, on each gene, thus keeping the strongest 95%. |
| cutoff.globalFilter | numeric; defines the cutoff for which sites to keep, the smallest step is 1% (0.01) in function pureClipGlobalFilter |
| est.bsResolution | character; level of resolution of the binding site width in function estimateBsWidth |
| est.geneResolution | character; level of resolution of the gene-wise filtering in function estimateBsWidth |
| est.maxBsWidth | numeric; the largest binding site width which should considered in the testing |
| est.minimumStepGain | numeric; the minimum additional gain in the score in percent the next binding site width has to have, to be selected as best option |
| est.maxSites | numeric; maximum number of PureCLIP sites that are used |
| est.subsetChromosome | character; define on which chromosome the estimation should be done in function estimateBsWidth |
| est.minWidth | the minimum size of regions that are subjected to the iterative merging routine, after the initial region concatenation. |
| est.offset | constant added to the flanking count in the signal-to-flank ratio calculation to avoid division by Zero |
| est.sensitive | logical; whether to enable sensitive pre-filtering before binding site merging or not |
| est.sensitive.size | numeric; the size (in nucleotides) of the merged sensitive region |
| est.sensitive.minWidth | numeric; the minimum size (in nucleoties) of the merged sensitive region |
| merge.minWidth | the minimum size of regions that are subjected to the iterative merging routine, after the initial region concatenation. |

`merge.minCrosslinks`
: the minimal number of positions to overlap with at least one crosslink event in the final binding sites

`merge.minClSites`
: the minimal number of crosslink sites that have to overlap a final binding site

`merge.CenterIsClSite`
: logical, whether the center of a final binding site must be covered by an initial crosslink site

`merge.CenterIsSummit`
: logical, whether the center of a final binding site must exhibit the highest number of crosslink events

`repro.cutoff`
: numeric; percentage cutoff to be used for the reproducibility quantile filtering

`repro.nReps`
: numeric; number of replicates that must meet the cutoff defined in `repro.cutoff` for a binding site to be called reproducible. Defaults to N-1.

`repro.minCrosslinks`
: numeric; minimal number of crosslinks a binding site needs to have to be called reproducible. Acts as a lower boundary for `repro.cutoff`. Defaults to 1.

`overlaps.geneWiseFilter`
: character; how overlaps should be handled in `pureClipGeneWiseFilter`

`overlaps.geneAssignment`
: character; how overlaps should be handled in `assignToGenes`

`overlaps.rule.geneAssignment`
: character vector; a vector of gene types that should be used to handle overlaps if option 'hierarchy' is selected for `assignToGenes`. The order of the vector is the order of the hierarchy.

`overlaps.TranscriptRegions`
: character; how overlaps should be handled in `assignToTranscriptRegions`

`overlaps.rule.TranscriptRegions`
: character vector; a vector of gene types that should be used to handle overlaps if option 'hierarchy' is selected for `assignToTranscriptRegions`. The order of the vector is the order of the hierarchy.

`stf.flank`
: character; how the flanking region shoule be set. Options are 'bs', 'manual'

`stf.flank.size`
: numeric; if flank='manual' provide the desired flanking size

`match.score`
: character; meta column name of the crosslink site

`match.geneID`
: character; meta column name of the genes

`match.geneName`
: character; meta column name of the gene name

`match.geneType`
: character; meta column name of the gene type

`match.ranges.score`
: a GRanges object, with numeric column for the score to match in function `annotateWithScore`

`match.option.score`
: character; meta column name of the crosslink site in function `annotateWithScore`

`anno.annoDB`
: an object of class `OrganismDbi` that contains the gene annotation !!! Experimental !!!

| | |
|---|---|
| anno.genes | an object of class GenomicRanges that represents the gene ranges directly |
| anno.transcriptRegionList | an object of class CompressedGRangesList that holds an ranges for each transcript region |
| quiet | logical; whether to print messages |
| veryQuiet | logical; whether to suppress all messages |
| ... | additional arguments passed to estimateBsWidth, makeBindingSites and reproducibilityFilter |

## Details

If only the annotation is provided (anno.genes and anno.transcriptRegionList), then binding sites size (bsSize) and gene-wise cutoff (cutoff.geneWiseFilter) are estimated using estimateBsWidth. To avoid this behavior one has to provide input values for the arguments bsSize and cutoff.geneWiseFilter.

If no binding site size is provided through bsSize, then estimateBsWidth is called to estimate the optimal size for the given data-set. The result of this estimation can be looked at with estimateBsWidthPlot and arguments can be adjusted if needed.

Use the processingStepsFlowChart function to get an overview of all steps carried out by the function.

For complete details on each step, see the manual pages of the respective functions. The BSFind function returns a BSFDataSet with ranges merged into binding sites. A full flowchart for the entire process can be visualized with processingStepsFlowChart. For each of the individual steps dedicated diagnostic plots exists. Further information can be found in our Bioconductor vignette: https://www.bioconductor.org/packages/release/bioc/html/BindingSiteFinder.html

## Value

an object of class BSFDataSet with ranges merged into binding sites given the inputs.

## See Also

BSFDataSet, estimateBsWidth, pureClipGlobalFilter, pureClipGeneWiseFilter, assignToGenes, assignToTranscriptRegions, annotateWithScore, reproducibilityFilter, calculateSignalToFlankScore, processingStepsFlowChart

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# Load genes
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
# load transcript regions
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
BSFind(object = bds, bsSize = 9, anno.genes = gns,
 anno.transcriptRegionList = regions, est.subsetChromosome = "chr22")
```

`calculateBsBackground` *Compute background coverage for binding sites per gene*

## Description

This function computes the background coverage used for the differential binding analysis to correct for transcript level changes. Essentially, the crosslink signal on each gene is split into crosslinks that can be attributed to the binding sites and all other signal that can be attributed to the background.

## Usage

```
calculateBsBackground(
  object,
  anno.annoDB = NULL,
  anno.genes = NULL,
  blacklist = NULL,
  use.offset = TRUE,
  ranges.offset = NULL,
  match.geneID.gene = "gene_id",
  match.geneID.bs = "geneID",
  match.geneID.blacklist = "geneID",
  generate.geneID.bs = FALSE,
  generate.geneID.blacklist = FALSE,
  uniqueID.gene = "gene_id",
  uniqueID.bs = "bsID",
  uniqueID.blacklist = "bsID",
  force.unequalSites = FALSE,
  quiet = FALSE,
  veryQuiet = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object with two conditions |
| `anno.annoDB` | an object of class `OrganismDbi` that contains the gene annotation. |
| `anno.genes` | an object of class `GenomicRanges` that represents the gene ranges directly |
| `blacklist` | GRanges; genomic ranges where the signal should be excluded from the background |
| `use.offset` | logical; if an offset region around the binding sites should be used on which the signal is excluded from the background |
| `ranges.offset` | numeric; number of nucleotides the offset window around each binding site should be wide (defaults to 1/2 binding site width - NULL) |
| `match.geneID.gene` | character; the name of the column with the gene ID in the genes meta columns used for matching binding sites to genes |

`match.geneID.bs`
: character; the name of the column with the gene ID in the binding sites meta columns used for matching binding sites to genes

`match.geneID.blacklist`
: character; the name of the column with the gene ID in the blacklist meta columns used for matching the blacklist regions with the genes

`generate.geneID.bs`
: logical; if the binding site to gene matching should be performed if no matching gene ID is provided

`generate.geneID.blacklist`
: logical; if the blacklist to gene matching should be performed if no matching gene ID is provided

`uniqueID.gene`
: character; column name of a unique ID for all genes

`uniqueID.bs`
: character; column name of a unique ID for all binding sites

`uniqueID.blacklist`
: character; column name of a unique ID for all blacklist regions

`force.unequalSites`
: logical; if binding sites of not identical width should be allowed or not

`quiet`
: logical; whether to print messages or not

`veryQuiet`
: logical; whether to print messages or not

`...`
: additional arguments; passed to `assignToGenes`

## Details

To avoid that crosslinks from binding sites contaminate the background counts a protective region around each binding sites can be spanned with `use.offset` the default width of the offset region is half of the binding site width, but can also be changed with the `ranges.offset` parameter.

Additional region that one wants to exclude from contributing to the background signal can be incorporated as `GRanges` objects through the `blacklist` option.

It is expected that binding sites are assigned to hosting genes prior to running this funciton (see `BSFind`). This means a unique gene ID is present in the meta columns of each binding site ranges. If this is not the case one can invoce the binding site to gene assignment with `generate.geneID.bs`. The same is true for the blacklist regions with option `generate.geneID.blacklist`.

It is expected that all binding sites are of the same size (See `BSFind` on how to achieve this). If this is however not the case and one wants to keep binding sites of different with then option `force.unequalSites` can be used.

This function is intended to be used for the generation of the count matrix used for the differential binding analysis. It is usually preceded by `combineBSF` and followed by `filterBsBackground`.

## Value

an object of class `BSFDataSet` with counts for binding sites, background and total gene added to the meta column of the ranges

## See Also

combineBSF, filterBsBackground

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make binding sites
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)

# change meta data
m = getMeta(bds)
m$condition = factor(c("WT", "WT", "KO", "KO"), levels = c("WT", "KO"))
bds = setMeta(bds, m)

# change signal
s = getSignal(bds)
names(s$signalPlus) = paste0(m$id, "_", m$condition)
names(s$signalMinus) = paste0(m$id, "_", m$condition)
bds = setSignal(bds, s)

# make example blacklist region
myBlacklist = getRanges(bds)
set.seed(1234)
myBlacklist = sample(myBlacklist, size = 500) + 4

# make background
bds.b1 = calculateBsBackground(bds, anno.genes = gns)

# make background - no offset
bds.b2 = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# make background - use blacklist
bds.b3 = calculateBsBackground(bds, anno.genes = gns, blacklist = myBlacklist)
```

---

# calculateBsFoldChange *Compute fold-changes per binding site*

---

## Description

Given count data for binding sites and background regions this function will compute fold-changes between two condition for each binding site. Computation is based on DESeq using the Likelihood ratio test to disentangle transcription level changes from binding site level changes.

## Usage

```
calculateBsFoldChange(
  object,
  fitType = "local",
  sfType = "ratio",
  minReplicatesForReplace = 10,
  independentFiltering = TRUE,
  alpha = 0.05,
  pAdjustMethod = "BH",
  minmu = 0.5,
  filterFun = NULL,
  use.lfc.shrinkage = FALSE,
  type = c("ashr", "apeglm", "normal"),
  svalue = FALSE,
  apeAdapt = TRUE,
  apeMethod = "nbinomCR",
  match.geneID = "geneID",
  quiet = TRUE,
  veryQuiet = FALSE,
  replaceNegative = FALSE,
  removeNA = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| fitType | either "parametric", "local", "mean", or "glmGamPoi" for the type of fitting of dispersions to the mean intensity. See DESeq for more details. |
| sfType | either "ratio", "poscounts", or "iterate" for the type of size factor estimation. See DESeq for more details. |
| minReplicatesForReplace | the minimum number of replicates required in order to use replaceOutliers on a sample. See DESeq for more details. |
| independentFiltering | logical, whether independent filtering should be applied automatically. See results for more details. |
| alpha | the significance cutoff used for optimizing the independent filtering. See results for more details. |
| pAdjustMethod | he method to use for adjusting p-values. See results for more details. |
| minmu | lower bound on the estimated count. See results for more details. |
| filterFun | an optional custom function for performing independent filtering and p-value adjustment. See results for more details. |
| use.lfc.shrinkage | logical; whether to compute shrunken log2 fold changes for the DESeq results. See lfcShrink for more details. |

| | |
|---|---|
| type | if 'ashr', 'apeglml' or 'normal' should be used for fold change shrinkage. See lfcShrink for more details. |
| svalue | logical, should p-values and adjusted p-values be replaced with s-values when using apeglm or ashr. See lfcShrink for more details. |
| apeAdapt | logical, should apeglm use the MLE estimates of LFC to adapt the prior. See lfcShrink for more details. |
| apeMethod | what method to run apeglm, which can differ in terms of speed. See lfcShrink for more details. |
| match.geneID | character; the name of the column with the gene ID in the binding sites meta columns used for matching binding sites to genes |
| quiet | logical; whether to print messages or not |
| veryQuiet | logical; whether to print messages or not |
| replaceNegative | logical; force negative counts to be replaces by 0. Be careful when using this, having negative counts can point towards problems with the gene annotation in use. |
| removeNA | logical; force binding sites with any NA value to be removed. Be careful when using this, having negative counts can point towards problems with the gene annotation in use. |

## Details

Fold-changes per binding sites are corrected for transcript level changes. Essentially, background counts are used to model transcript level changes, which are then used to compute fold-changes per binding site, which are corrected for the observed transcript level changes. This is done by using a Likelihood ratio test comparing the full model (~condition + type + condition:type) to the reduced model (~condition + type).

Fold-changes for the transcript level changes are modeled explicitly in a second round of the DESeq workflow, using the default Wald test to compare changes between the conditions (~condition). Counts attributed to binding sites are removed from the gene level counts.

Results from both calculation rounds can be filtered and further manipulated with parameters given in the DESeq2 framework (see results, lfcShrink).

This function is intended to be used right after a call of filterBsBackground.

## Value

a BSFDataSet object, with results from the DESeq analysis added to the meta columns of the binding site ranges.

## See Also

calculateBsBackground, filterBsBackground, plotBsBackgroundFilter, DESeq

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make example dataset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns)

# calculate fold changes - no shrinkage
bds = calculateBsFoldChange(bds)

# calculate fold changes - with shrinkage
bds = calculateBsFoldChange(bds, use.lfc.shrinkage = TRUE)
```

---

# calculateSignalToFlankScore

*Calculate signal-to-flank score*

---

## Description

This function calculates the signal-to-flank ratio for all present binding sites.

## Usage

```
calculateSignalToFlankScore(
  object,
  flank = c("bs", "manual"),
  flank.size = NULL,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| flank | character; how the flanking region shoule be set. Options are 'bs', 'manual' |
| flank.size | numeric; if flank='manual' provide the desired flanking size |
| quiet | logical; whether to print messages |

## Details

Each input range is treated as a binding site. For a particular binding site all overlapping crosslinks are summed up and divided by the normalized sum of the crosslinks in the two adjecent regions of the same size. This is done for all bining sites and the ratio is reported as a score.

The function is part of the standard workflow performed by BSFind.

## Value

an object of class BSFDataSet with signal-to-flank ratios added to the meta column of the ranges.

## See Also

BSFind, bindingSiteDefinednessPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds = makeBindingSites(bds, bsSize = 5)
bds = calculateSignalToFlankScore(bds)
```

---

# clipCoverage *Coverage function for BSFDataSet objects*

---

## Description

Function that computes a crosslink coverage with all samples over all ranges given in the BSFDataSet. The coverage can be summarized over all combinations of the three dimension (samples, ranges, positions).

## Usage

```
clipCoverage(
  object,
  ranges.merge = FALSE,
  ranges.merge.method = c("sum", "mean"),
  positions.merge = FALSE,
  positions.merge.method = c("sum", "mean"),
  samples.merge = TRUE,
  samples.group = c("all", "condition"),
  samples.merge.method = c("sum", "mean"),
  out.format = c("granges", "data.frame"),
  out.format.overwrite = FALSE,
  match.rangeID = "bsID",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| ranges.merge | logical; whether to merge ranges |

`ranges.merge.method`
: character; how to combine ranges ('sum' or 'mean')

`positions.merge`
: logical; whether to merge positions

`positions.merge.method`
: character; how to combine positions ('sum' or 'mean')

`samples.merge`
: logical: whether to merge samples

`samples.group`
: character; how samples should be grouped when combining ('all', 'condition')

`samples.merge.method`
: charater; how to combine positions ('sum' or 'mean')

`out.format`
: character; how the coverage should be returned ('data.frame' or 'granges'). Note that option 'granges' only exists if the output coverage is of the same rows as the input ranges.

`out.format.overwrite`
: logical; if `out.format='granges'`, then decide wheter the meta columns should be extended by the coverage information or be overwritten

`match.rangeID`
: character; unique internal identifier. Name of the meta column of the input ranges that should be used as identifier to match the coverage back to the input ranges. Is 'bsID' as default, since that ID exists for all binding sites after `makeBindingSites` was called.

`quiet`
: logical; whether to print messages

## Details

When summarizing the crosslink coverage over samples (`samples.merge=TRUE`) one can decide whether to summarize all samples or whether to keep conditions separate (`samples.group`). This either reduces the samples dimension to a single matrix, or a list. For a binding site set with 100 binding sites of width=7 and 4 replicates with 2 conditions, the following options are possible. With merging enabled and `samples.group='all'` the coverage of all samples is combined. With `samples.group='condition'` only samples of the same condition are grouped.

When summarizing the crosslink coverage over ranges, all ranges are combined which reduces the ranges dimension to a single vector. This turns eg. a binding site set of 100 binding sites with width=7 into a vector of length 100 with exactly one column. Depending on how the samples were summarized, the result can be a single such vector, or a list.

When summarizing the crosslink coverage over positions, all positions are combined which reduces the positions dimension to a single vector. This turns eq. a binding site set of 100 binding sites with width=7 into a vector of length 1 with 7 columns. Depending on how the samples were summarized, the result can be a single such vector, or a list.

For all summarizing operations options `sum` and `mean` exists. This allows for normalization by the eg. the number of binding sites, size of the range, number of sample, etc..

If the resulting object does have a dimension that fits to the number of input ranges the result can be directly attached to them. Basically extending the `GRanges` object (`out.format`).

## Value

an object of class specified in `out.format`

## See Also

BSFDataSet, BSFind

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bds = makeBindingSites(bds, bsSize = 7)
# sum of each replicate over each binding site position
c1 = clipCoverage(bds, out.format = "data.frame", positions.merge = TRUE,
 ranges.merge = FALSE, samples.merge = FALSE)
# total signal per binding site from all samples
c2 = clipCoverage(bds, out.format = "granges", positions.merge = FALSE,
 ranges.merge = TRUE, samples.merge = TRUE, samples.group = "all")
# total signal per binding site from all samples - split by condition
c3 = clipCoverage(bds, out.format = "granges", positions.merge = FALSE,
 ranges.merge = TRUE, samples.merge = TRUE, samples.group = "condition")
```

---

# collapseReplicates *Collapse signal from replicates*

---

## Description

Collapses all replicates merges all samples from a BSFDataSet object into a single signal stream, only split by minus and plus strand.

## Usage

```
collapseReplicates(object, collapseAll = FALSE)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| collapseAll | TRUE/FALSE, if all samples should be collapsed (TRUE), or if samples should be kept separate by condition (FALSE) |

## Value

object of type BSFDataSet with updated signal

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bdsNew = collapseReplicates(bds)
```

---

combineBSF *Combine multiple* BSFDataSet *objects*

---

## Description

This function combines all BSFDataSet objects from the input list into a single BSFDataSet object.

## Usage

```
combineBSF(
  list,
  overlaps.fix = TRUE,
  combine.bsSize = NULL,
  combine.name = NULL,
  force.reload = FALSE,
  quiet = TRUE,
  veryQuiet = FALSE
)
```

## Arguments

| | |
|---|---|
| list | list; a list of objects from class BSFDataSet that should be combined |
| overlaps.fix | logical; if partially overlapping binding sites should be re-centered |
| combine.bsSize | numeric; the binding site size that the merged sites should have. Default=NULL, then bsSize is taken from the input objects in list. |
| combine.name | character; meta table name of the combined object. Default=NULL; then name is set to 'combined' |
| force.reload | logical; whether the signal should be derived from the merge of the input objects given in list or if the signal should be re-loaded from the path given in the meta data. |
| quiet | logical; whether to print messages or not |
| veryQuiet | logical; whether to print status messages or not |

## Details

Meta-data tables are added to each other by performing a row-wise bind, basically adding all meta data tables underneath each other.

The default way of signal combination is merging all signal lists on the level of the indivdual samples. One can also force a re-load of the signal list component by using `force.reload=TRUE`. The signal can be combined by

The ranges are combined by adding both granges objects together. With option `overlaps.fix` one can decide if partially overlapping ranges should be combined into a single range or not. If this option is FALSE one is likely to have overlapping binding sites after the merge. If this option is TRUE, then the combined coverage is used to guide the new center point for these cases.

The `combine.bsSize` option allows one to set a unique bsSize for all objects that should be combined. Although it is recommended to combine only objects with the same bsSize this option can be used to ensure that the merged result has the same bsSize for all ranges.

This function is usually used to combine two datasets in the context of a differntial testing analysis.

## Value

an object of class BSFDataSet with ranges, signal and meta data resulting from the merge of the input objects.

## See Also

processingStepsFlowChart, calculateBsBackground

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# make binding sites
bds = makeBindingSites(bds, bsSize = 7)

# split ranges in two groups
allRanges = getRanges(bds)
set.seed(1234)
idx = sample(1:length(allRanges), size = length(allRanges)/2, replace = FALSE)
r1 = allRanges[idx]
r2 = allRanges[-idx]

# splite meta data
allMeta = getMeta(bds)
m1 = allMeta[1:2,]
m2 = allMeta[3:4,]

# create new objects
bds1 = setRanges(bds, r1)
bds2 = setRanges(bds, r2)
bds1 = setMeta(bds1, m1)
```

```
bds2 = setMeta(bds2, m2)
bds1 = setName(bds1, "test1")
bds2 = setName(bds2, "test2")

# merge two objects with '+' operator
c1 = bds1 + bds2

# merge two objects from list
list = list(bds1, bds2)
c1 = combineBSF(list = list, overlaps.fix = TRUE,
 combine.bsSize = NULL, combine.name = NULL, quiet = TRUE)
```

---

`coverageOverRanges` *Coverage function for BSFDataSet objects*

---

## Description

The crosslink coverage is computed for all ranges in the the given `BSFDataSet` object (see `BSFDataSet` for details). Depending on the `returnOptions` the resulting coverage information is summarized, suitable for diverse computation and plotting tasks. The coverage can only be compute for objects with identical ranges.

## Usage

```
coverageOverRanges(
  object,
 returnOptions = c("merge_ranges_keep_positions", "merge_replicates_per_condition",
    "merge_all_replicates", "merge_positions_keep_replicates"),
  method = "sum",
  allowNA = FALSE,
  quiet = TRUE
)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `returnOptions` | one of merge_ranges_keep_positions, merge_replicates_per_condition, merge_all_replicates, merge_positions_keep_replicates |
| `method` | sum/ mean, select how replicates/ ranges should be summarized |
| `allowNA` | TRUE/ FALSE, allow NA values in the output if input ranges are of different width |
| `quiet` | logical, whether to print messages |

## Details

If `returnOptions` is set to merge_ranges_keep_positions: Returns a matrix with ncol being the nucleotides of the ranges (equal to the width of the input ranges) and nrow being the number of replicates in the meta information.

If `returnOptions` is set to merge_replicates_per_condition: Returns a list of matrices. Each list corresponds to one condition set in the meta information. The matrix in each entry has ncols equal to the ranges width and nrow equal to the number of ranges. Counts per ranges and position are summed.

If `returnOptions` is set to merge_all_replicates: Returns a matrix with ncols equal to the range width and nrow equal to the number of ranges. Counts per range and position are summed.

If `returnOptions` is set to merge_positions_keep_replicates: Returns a GRanges object where the counts are summed for each replicate and added to the original granges object.

## Value

an object of class specified in `returnOptions`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

rng = coverageOverRanges(bds, returnOptions = "merge_ranges_keep_positions")
rng = coverageOverRanges(bds, returnOptions = "merge_replicates_per_condition")
rng = coverageOverRanges(bds, returnOptions = "merge_all_replicates")
rng = coverageOverRanges(bds, returnOptions = "merge_positions_keep_replicates")
```

---

# duplicatedSitesPlot *Plot the number of overlaps when assigning crosslink sites to genes*

---

## Description

A diagnostic function that plots the number of crosslink sites with their respective overlapping rate. The function `pureClipGeneWiseFilter` is expected to be executed prior to calling this plot function.

## Usage

```
duplicatedSitesPlot(object)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object |

## Value

a plot of type ggplot

## See Also

pureClipGeneWiseFilter

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# Load GRanges with genes
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
# apply 5% gene-wise filter
bds = pureClipGeneWiseFilter(object = bds, anno.genes = gns, cutoff = 0.5,
 overlaps = "keepSingle")
duplicatedSitesPlot(bds)
```

---

# estimateBsWidth *Function to estimate the appropriate binding site width together with the optimal gene-wise filter level.*

---

## Description

This function tests different width of binding sites for different gene-wise filtering steps. For each test the signal-to-score ratio is calculated. The mean over all gene-wise filterings at each binding site width is used to extract the optimal width, which serves as anchor to select the optimal gene-wise filter.

## Usage

```
estimateBsWidth(
  object,
  bsResolution = c("medium", "fine", "coarse"),
  geneResolution = c("medium", "coarse", "fine", "finest"),
  est.maxBsWidth = 13,
  est.minimumStepGain = 0.02,
  est.maxSites = Inf,
  est.subsetChromosome = "chr1",
  est.minWidth = 2,
  est.offset = 1,
  sensitive = FALSE,
  sensitive.size = 5,
  sensitive.minWidth = 2,
  anno.annoDB = NULL,
  anno.genes = NULL,
```

```
  bsResolution.steps = NULL,
  geneResolution.steps = NULL,
  quiet = TRUE,
  veryQuiet = FALSE,
  reportScoresPerBindingSite = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored crosslink sites. This means that ranges should have a width = 1. |
| bsResolution | character; level of resolution at which different binding site width should be tested |
| geneResolution | character; level of resolution at which gene-wise filtering steps should be tested |
| est.maxBsWidth | numeric; the largest binding site width which should considered in the testing |
| est.minimumStepGain | numeric; the minimum additional gain in the score in percent the next binding site width has to have, to be selected as best option |
| est.maxSites | numeric; maximum number of PureCLIP sites that are used; |
| est.subsetChromosome | character; define on which chromosome the estimation should be done in function estimateBsWidth |
| est.minWidth | the minimum size of regions that are subjected to the iterative merging routine, after the initial region concatenation. |
| est.offset | constant added to the flanking count in the signal-to-flank ratio calculation to avoid division by Zero |
| sensitive | logical; whether to enable sensitive pre-filtering before binding site merging or not |
| sensitive.size | numeric; the size (in nucleotides) of the merged sensitive region |
| sensitive.minWidth | numeric; the minimum size (in nucleoties) of the merged sensitive region |
| anno.annoDB | an object of class OrganismDbi that contains the gene annotation (!!! Experimental !!!). |
| anno.genes | an object of class GenomicRanges that represents the gene ranges directly |
| bsResolution.steps | numeric vector; option to use a user defined threshold for binding site width directly. Overwrites bsResolution |
| geneResolution.steps | numeric vector; option to use a user defined threshold vector for gene-wise filtering resolution. Overwrites geneResolution |
| quiet | logical; whether to print messages |
| veryQuiet | logical; whether to suppress all messages |

`reportScoresPerBindingSite`
: report the ratio score for each binding site separately. Warning! This is for debugging and testing only. Downstream functions can be impaired.

`...`
: additional arguments passed to `pureClipGeneWiseFilter`

## Details

Parameter estimation is done on a subset of all crosslink sites (`est.subsetChromosome`).

Gene-level filter can be tested with varying levels of accuracy ranging from 'finest' to 'coarse', representing 1 20

Binding site computation at each step can be done on three different accuracy level (`bsResolution`). Option 'fine' is equal to a normal run of the `makeBindingSites` function. 'medium' will perform a shorter version of the binding site computation, skipping some of the refinement steps. Option 'coarse' will approximate binding sites by merged crosslinks regions, aligning the center at the site with the highest score.

For each binding site in each set given the defined resolutions a signal-to- flank score ratio is calculated and the mean of this score per set is returned. Next a mean of means is created which results in a single score for each binding site width that was tested. The width that yielded the highest score is selected as optimal. In addtion the `minimumStepGain` option allows control over the minimum additional gain in the score that a tested width has to have to be selected as the best option.

To enhance the sensitivity of the binding site estimation, the sensitivity mode exists. In this mode crosslink sites undergo a pre-filtering and merging step, to exclude potential artifical peaks (experimental-, mapping-biases). If sensitivity mode is activated the `est.minWidth` option should be set to 1.

The optimal geneFilter is selected as the first one that passes the merged mean of the selected optimal binding site width.

The function is part of the standard workflow performed by `BSFind`.

## Value

an object of class `BSFDataSet` with binding sites with the 'params' slots 'bsSize' and 'geneFilter' being filled

## See Also

`BSFind`, `estimateBsWidthPlot`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
estimateBsWidth(bds, anno.genes = gns, est.maxBsWidth = 19,
 geneResolution = "coarse", bsResolution = "coarse", est.subsetChromosome = "chr22")
```

## estimateBsWidthPlot *Plot the signal-to-flank score for varying gene-wise filter and binding site width*

### Description

A diagnostic function that plots the the signal-to-flank score as a mean for each binding site width and gene-wise filter as indicated when executing estimateBsWidth. Additionally a mean of means visualizes the overall trend and a red line indicates the suggested optimal binding site width. The function estimateBsWidth is expected to be executed prior to calling this plot function.

### Usage

```
estimateBsWidthPlot(object)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |

### Value

a plot of type ggplot

### See Also

estimateBsWidth

### Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
bds = estimateBsWidth(bds, anno.genes = gns, est.maxBsWidth = 19,
 geneResolution = "coarse", bsResolution = "coarse", est.subsetChromosome = "chr22")
estimateBsWidthPlot(bds)
```

## exportTargetGenes *Function to export sorted RBP target genes*

### Description

Genes with binding sites are target genes of the RBP. They can be exported as 'csv' or 'xlsx' file. Genes can be sorted by the sum of the individual binding sites score, or by the number of binding sites per gene.

## Usage

```
exportTargetGenes(
  object,
  path = "./",
  format = c("csv", "xslx"),
  sort = c("score", "bs"),
  split = c("none", "geneType", "transcriptRegion")
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored ranges |
| path | A path to where the output should be stored |
| format | output file format |
| sort | sorting rule for genes |
| split | if and how the output file should be split |

## Details

As output option, one can either output all genes in a single file, or split by either gene-type or transcript-region. This options requires that either BSFind or the individual functions assignToGenes, and assignToTranscriptRegions were run.

## Value

a file of the type specified in format

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
## Not run:
# export
# exportTargetGenes(bds)

## End(Not run)
```

---

# exportToBED *Wrapper function to export binding sites as BED files*

---

## Description

Function that serves as a wrapper function for rtracklayer::export.

### Usage

```
exportToBED(object, con)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored ranges |
| con | A path or URL |

### Value

a .bed file

### See Also

BSFind

### Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
## Not run:
# export
# exportToBED(bds, con = "./myfile.bed")

## End(Not run)
```

---

## filterBsBackground *Filter for genes not suitable for differential testing*

---

### Description

This function removes genes where the differential testing protocol can not be applied to, using count coverage information on the binding sites and background regions per gene, through the following steps:

1. Remove genes with overall not enough crosslinks: `minCounts`
2. Remove genes with a disproportion of counts in binding sites vs. the background: `balanceBackground`
3. Remove genes where the expression between conditions is too much off balance: `balanceCondition`

## Usage

```
filterBsBackground(
  object,
  minCounts = TRUE,
  minCounts.cutoff = 100,
  balanceBackground = TRUE,
  balanceBackground.cutoff.bs = 0.3,
  balanceBackground.cutoff.bg = 0.7,
  balanceCondition = TRUE,
  balanceCondition.cutoff = 0.05,
  match.geneID = "geneID",
  flag = FALSE,
  quiet = FALSE,
  veryQuiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object with computed count data for binding sites and background regions |
| `minCounts` | logical; whether to use the minimum count filter |
| `minCounts.cutoff` | numeric; the minimal number of crosslink per gene over all samples for the gene to be retained (default = 100) |
| `balanceBackground` | logical; whether to use the counts balancing filter between binding sites and background |
| `balanceBackground.cutoff.bs` | numeric; the maximum fraction of the total signal per gene that can be within binding sites (default = 0.2) |
| `balanceBackground.cutoff.bg` | numeric; the minimum fraction of the total signal per gene that can be within the background (default = 0.8) |
| `balanceCondition` | logical; whether to use the counts balancing filter between conditions |
| `balanceCondition.cutoff` | numeric; the maximum fraction of the total signal that can be attributed to only one condition |
| `match.geneID` | character; the name of the column with the gene ID in the binding sites meta columns used for matching binding sites to genes |
| `flag` | logical; whether to remove or flag binding sites from genes that do not pass any of the filters |
| `quiet` | logical; whether to print messages or not |
| `veryQuiet` | logical; whether to print messages or not |

## Details

To remove genes with overall not enough crosslinks (`minCounts`) all counts are summed up per gene across all samples and compared to the minimal count threshold (`minCounts.cutoff`).

To remove genes with a count disproportion between binding sites and background regions crosslinks are summed up for binding sites and background per gene. These sums are combined in a ratio. Genes where eg. 50% of all counts are within binding sites would be removed (see `balanceBackground.cutoff.bs` and `balanceBackground.cutoff.bg`).

To remove genes with very large expression differences between conditions, crosslinks are summed up per gene for each condition. If now eg. the total number of crosslinks is for 98% in one condition and only 2% of the combined signal is in the second condition, expression levels are too different for a reliable comparisson (see `balanceCondition.cutoff`).

This function is intended to be used right after a call of `calculateBsBackground`.

## Value

an object of class `BSFDataSet` with biniding sites filtered or flagged by the above filter options

## See Also

`calculateBsBackground`, `plotBsBackgroundFilter`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make testset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# use all filters and remove binding sites that fail (default settings)
f0 = filterBsBackground(bds)

# do not use the condition balancing filter
f1 = filterBsBackground(bds, balanceCondition = FALSE)

# use only the minimum count filter and flag binding sites instead of
# removing them
f3 = filterBsBackground(bds, flag = TRUE, balanceCondition = FALSE,
 balanceBackground = FALSE)
```

geneOverlapsPlot *UpSet-plot to that shows the gene type overlaps*

## Description

A diagnostic function that plots the gene types of binding sites on overlapping loci genes. The function assignToGenes is expected to be executed prior to calling this plot function.

## Usage

```
geneOverlapsPlot(object, text.size = NULL, show.title = TRUE)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| text.size | numeric; fontsize of all numbers on axis |
| show.title | logical; if plot title should be visible |

## Value

a plot of type ggplot

## See Also

assignToGenes targetGeneSpectrumPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# Load GRanges with genes
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
bds = makeBindingSites(object = bds, bsSize = 9)
bds = assignToGenes(bds, anno.genes = gns)
geneOverlapsPlot(bds)
```

geneRegulationPlot *Gene Regulation Plot*

## Description

Display the fold-change of all binding sites from a given gene on a relative per-nucleotide scale. Binding sites are displayed as dots and with increasing log2 fold-change, they deviate stronger from the center line.

## Usage

```
geneRegulationPlot(
  object,
  plot.geneID = NULL,
  anno.annoDB = NULL,
  anno.genes = NULL,
  match.geneID = "gene_id",
  match.geneName = "gene_name",
  plot.gene.n.tiles = 100,
  alpha = 0.05,
  lfc.cutoff = 2,
  transcript.regions.outlier.handle = c("first", "second", "both", "remove"),
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | object; a BSFDataSet object |
| plot.geneID | character; the gene id of the gene to display. The id must match with the gene ids given in the annotation object. |
| anno.annoDB | object; an object of class OrganismDbi that contains the gene annotation (!!! Experimental !!!). |
| anno.genes | object; an object of class GenomicRanges that represents the gene ranges directly. |
| match.geneID | character; meta column name of the gene ID |
| match.geneName | character; meta column name of the gene name |
| plot.gene.n.tiles | numeric; number of tiles the gene should be split in |
| alpha | numeric; the alpha value to show significantly regulated binding sites. This should match the alpha value used in calculateBsFoldChange. |
| lfc.cutoff | numeric; log2 fold-change cutoff to show significantly regulated binding sites. This should match the lfc.cutoff value used in calculateBsFoldChange. |
| transcript.regions.outlier.handle | character; the option how to handle multiple transcript region annotations being present for the same binding site. |
| quiet | logical; whether to print messages |

## Details

For this function to work, binding sites must be assigned to hosting genes using assignToGenes. It is also recommended to assing binding sites to transcript regions with assignToTranscriptRegions.

It is also necessary to calculate the log2 fold-change of binding sites between two conditions using the differential binding workflow calculateBsFoldChange.

If in addition the transcript regions of the binding sites are given, then shapes are changed accordingly. An edge case can arise from the merging of two BSFDataSet objects. If binding sites are overlapping and slightly offset close to the end of a particular transcript region annotation, they might be assigned to different regions in both objects. This results in some ambiguity after the merge, where for instance a binding site can be assigned to CDS and 3'UTR. To handle how such edge cases are displayed, the `transcript.regions.outlier.handle` exists. As default, simply the region of the object that was merged first is shown. If one is interested in showing all regions, then the options `both` displays both annotations at the same time and labels them accordingly.

## Value

an object of class `ggplot2`

## See Also

BSFind, calculateBsFoldChange assignToGenes assignToTranscriptRegions

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])

# make testset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = assignToTranscriptRegions(object = bds, anno.transcriptRegionList = regions)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# use all filters and remove binding sites that fail (default settings)
bds = filterBsBackground(bds)

# calculate fold-changes
bds = calculateBsFoldChange(bds)

# make example plot
exampleGeneId = "ENSG00000253352.10"
geneRegulationPlot(bds, plot.geneID = exampleGeneId, anno.genes = gns)
```

## getMeta *Accessor method for the meta data of the BSFDataSet object*

### Description

Meta data is stored as a `data.frame` and must contain the columns "condition", "clPlus" and "clMinus".

### Usage

```
getMeta(object)

## S4 method for signature 'BSFDataSet'
getMeta(object)
```

### Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |

### Value

returns the meta data `data.frame` with the columns "condition", "clPlus" and "clMinus".

### See Also

BSFDataSet

### Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

getMeta(bds)
```

## getName *Accessor method for the name of the BSFDataSet object*

### Description

The name slot holds the name of the dataset

### Usage

```
getName(object)

## S4 method for signature 'BSFDataSet'
getName(object)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |

### Value

returns the name of the dataset

### See Also

BSFDataSet

### Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

getName(bds)
```

---

## getRanges *Accessor method for the ranges of the BSFDataSet object*

---

### Description

The ranges slot holds the genomic ranges information of the sites currently in the object. They are encoded as a GRanges object with each binding site having a single ranges entry.

### Usage

```
getRanges(object)

## S4 method for signature 'BSFDataSet'
getRanges(object)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |

### Value

returns the genomic ranges (GRanges) of the associated ranges

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

getRanges(bds)
```

---

getSignal *Accessor method for the signal data of the BSFDataSet object*

---

## Description

Signal data is loaded from the path specified in getMeta columns "clPlus" and "clMinus" and stored as a list of RLE lists.

## Usage

```
getSignal(object)

## S4 method for signature 'BSFDataSet'
getSignal(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |

## Value

returns the signal data, as list of RLE list for each strand, named after the meta data columns "clPlus" and "clMinus"

## See Also

getMeta BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

getSignal(bds)
```

getSummary *Accessor method for the summary slot of the BSFDataSet object*

## Description

The summary slot is used to track information of the filtering steps applied in the makeBindingSites function

## Usage

```
getSummary(object, ...)

## S4 method for signature 'BSFDataSet'
getSummary(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| ... | additional arguments |

## Value

returns the summary information storted in the summary slot after makeBindingSites was run

## See Also

BSFDataSet makeBindingSites

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1)

getSummary(bds)
```

globalScorePlot *Plot the PureCLIP score distribution after re-assignment*

## Description

A diagnostic function that plots the PureCLIP score distribution on a log2 scale after the re-assignment on binding site level. The function annotateWithScore is expected to be executed prior to calling this plot function.

## Usage

```
globalScorePlot(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |

## Value

a plot of type ggplot

## See Also

annotateWithScore

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds1 = makeBindingSites(object = bds, bsSize = 9)
bds1 = annotateWithScore(bds1, match.ranges = getRanges(bds))
globalScorePlot(bds1)
```

imputeBsDifferencesForTestdata
*Impute artificial differences in the example data set*

## Description

A function that works only on the test data set provided with the package. It is used for internal testing and the making of examples to showcase the differential binding functions.

## Usage

```
imputeBsDifferencesForTestdata(object, size = 5, change.per = 0.1)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object; explicitly the test data set from the extdata folder |
| size | numeric; the number of positions on which signal should be deleted, counting from the start |
| change.per | numeric; the percentage of ranges that should be effected by the change. |

### Details

Differences between samples are artificially introduced by removing the signal on a random set of binding sites of the input.

### Value

object a BSFDataSet object with the signal slot adapted to reflect changes in binding between two artificial conditions

### Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)

bds = imputeBsDifferencesForTestdata(bds)
```

---

## makeBindingSites *Define equally sized binding sites from peak calling results and iCLIP crosslink events.*

---

### Description

This function performs the merging of single nucleotide crosslink sites into binding sites of a user defined width (bsSize). Depending on the desired output width crosslink sites with a distance closer than bsSize -1 are concatenated. Initially all input regions are concatenated and then imperatively merged and extended. Concatenated regions smaller than minWidth are removed prior to the merge and extension routine. This prevents outlier crosslink pileup, eg. mapping artifacts to be integrated into the final binding sites. All remaining regions are further processed and regions larger than the desired output width are interactively split up by setting always the position with the highest number of crosslinks as center. Regions smaller than the desired width are symmetrically extended. Resulting binding sites are then filtered by the defined constraints.

## Usage

```
makeBindingSites(
  object,
  bsSize = NULL,
  minWidth = 2,
  minCrosslinks = 2,
  minClSites = 1,
  centerIsClSite = TRUE,
  centerIsSummit = TRUE,
  sub.chr = NA,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object (see BSFDataSet) |
| bsSize | an odd integer value specifying the size of the output binding sites |
| minWidth | the minimum size of regions that are subjected to the iterative merging routine, after the initial region concatenation. |
| minCrosslinks | the minimal number of positions to overlap with at least one crosslink event in the final binding sites |
| minClSites | the minimal number of crosslink sites that have to overlap a final binding site |
| centerIsClSite | logical, whether the center of a final binding site must be covered by an initial crosslink site |
| centerIsSummit | logical, whether the center of a final binding site must exhibit the highest number of crosslink events |
| sub.chr | chromosome identifier (eg, chr1, chr2) used for subsetting the BSFDataSet object. This option can be used for testing different parameter options |
| quiet | logical, whether to print info messages |

## Details

The bsSize argument defines the final output width of the merged binding sites. It has to be an odd number, to ensure that a binding site has a distinct center.

The minWidth parameter is used to describe the minimum width a ranges has to be after the initial concatenation step. For example: Consider bsSize = 9 and minWidth = 3. Then all initial crosslink sites that are closer to each other than 8 nucleotides (bsSize -1) will be concatenated. Any of these ranges with less than 3 nucleotides of width will be removed, which reflects about 1/3 of the desired binding site width.

The argument minCrosslinks defines how many positions of the binding sites are covered with at least one crosslink event. This threshold has to be defined in conjunction with the binding site width. A default value of 3 with a binding site width of 9 means that 1/3 of all positions in the final binding site must be covered by a crosslink event. Setting this filter to 0 deactivates it.

The minClSites argument defines how many positions of the binding site must have been covered by the original crosslink site input. If the input was based on the single nucleotide crosslink positions computed by PureCLIP than this filter checks for the number of positions originally identified

by PureCLIP in the computed binding sites. The default of `minClSites` = 1 essentially deactivates this filter. Setting this filter to 0 deactivates it.

The options `centerIsClSite` and `centerIsSummit` ensure that the center of each binding site is covered by an initial crosslink site and represents the summit of crosslink events in the binding site, respectively.

The option `sub.chr` allows to run the binding site merging on a smaller subset (eg. "chr1") for improoved computational speed when testing the effect of various binding site width and filtering options.

## Value

an object of type BSFDataSet with modified ranges

## See Also

BSFDataSet, BSFind, mergeCrosslinkDiagnosticsPlot, makeBsSummaryPlot

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# standard options, no subsetting
bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1)

# standard options, with subsetting
bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1, sub.chr = "chr22")
```

---

# makeBsSummaryPlot *Plot binding site filter diagnostics*

---

## Description

A diagnostic function that plots the number of binding sites retained after each filtering step calculated in makeBindingSites. The function makeBindingSites is expected to be executed prior to calling this plot function.

## Usage

```
makeBsSummaryPlot(object)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |

## Value

a plot of type ggplot

## See Also

makeBindingSites

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds = makeBindingSites(object = bds, bsSize = 9)
makeBsSummaryPlot(bds)
```

---

# mergeCrosslinkDiagnosticsPlot

*Plot binding site merging diagnostics*

---

## Description

A diagnostic function that plots the number of regions to merge over the width of these regions for each merging iteration calculated in makeBindingSites. The function makeBindingSites is expected to be executed prior to calling this plot function.

## Usage

```
mergeCrosslinkDiagnosticsPlot(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |

## Value

a plot of type ggplot

## See Also

makeBindingSites

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds = makeBindingSites(object = bds, bsSize = 9)
mergeCrosslinkDiagnosticsPlot(bds)
```

`mergeSummaryPlot` *Plot summarized results of the different binding site merging and filtering steps*

## Description

Bar charts produced for the different filter steps in the binding site merging routine. Depending on the selected option (`select`) all or only a user defined filter can be shown.

## Usage

```
mergeSummaryPlot(
  object,
  select = c("all", "filter", "inputRanges", "minClSites", "mergeCrosslinkSites",
    "minCrosslinks", "centerIsClSite", "centerIsSummit"),
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataObject, with the makeBindingSites function already run |
| `select` | one of "all", "filter", "inputRanges", "minCLSites", "mergeCrosslinkSites", "minCrosslinks", "centerIsClSite" or "centerIsSummit". Defines which parameter is selected for plotting. |
| `...` | further arguments passed to ggplot |

## Details

If `object` is a single BSFDataObject a single coverage plot will be drawn, whereas if it is a list of BSFDataObjects, then faceting is used to make a plot for each list element.

## Value

a plot of type ggplot after the `makeBindingSites` function was run

## See Also

`makeBindingSites`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# plotting a single object
bds0 <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1)
```

```
mergeSummaryPlot(bds0)

# plotting mulitple obejcts
bds1 <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1, sub.chr = "chr22")
bds2 <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 3, sub.chr = "chr22")
l = list(`1. bsSize = 3` = bds1, `2. bsSize = 9` = bds2)
mergeSummaryPlot(l, width = 20)
```

---

`plotBsBackgroundFilter` *Diagnostic plots for the differential binding background*

---

## Description

To perform differential binding analysis between two conditions the `calculateBsBackground` function groups crosslinks per gene into those from binding sites and those from background regions. The `filterBsBackground` function perfroms certain filtering operations on that background to ensure that it's suitable for differential testing. This function visually displays the effect of these filtering operations.

## Usage

```
plotBsBackgroundFilter(
  object,
  filter = c("minCounts", "balanceBackground", "balanceCondition")
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object with background counts filtered by `filterBsBackground` |
| `filter` | character; which filter to display in the plot (one of: 'minCounts', 'balanceBackground', 'balanceCondition') |

## Value

a plot of type `ggplot`

## See Also

`calculateBsBackground`

`filterBsBackground`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make testset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# use all filters and remove binding sites that fail (default settings)
bds = filterBsBackground(bds)

# display minCount filter
plotBsBackgroundFilter(bds, filter = "minCounts")

# display balance background filter
plotBsBackgroundFilter(bds, filter = "balanceBackground")

# display balance condition filter
plotBsBackgroundFilter(bds, filter = "balanceCondition")
```

---

# plotBsMA *MA style plot*

---

## Description

Wrapper that plots differential binding results as MA plot. For each binding site the estimated baseMean (log2) is shown on X and the fold-change (log2) is shown on Y.

## Usage

```
plotBsMA(object, what = c("bs", "bg"), sig.threshold = 0.05)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with results calculated by calculateBsFoldChange |
| what | character; whether to show results for binding sites or the background (one of: 'bs', 'bg') |
| sig.threshold | numeric; what P value significance level to use (default = 0.05) |

## Value

a plot of type ggplot

## See Also

calculateBsFoldChange

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make testset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# use all filters and remove binding sites that fail (default settings)
bds = filterBsBackground(bds)

# calculate fold-changes
bds = calculateBsFoldChange(bds)

# make MA plot
plotBsMA(bds)
```

---

# plotBsVolcano *Volcano style plot*

---

## Description

Wrapper that plots differential binding results as volcano plot. For each binding site the estimated fold-change (log2) is shown on X and the adjusted P value (-log10) is shown on Y.

## Usage

```
plotBsVolcano(object, what = c("bs", "bg"), sig.threshold = 0.05)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with results calculated by calculateBsFoldChange |
| what | character; whether to show results for binding sites or the background (one of: 'bs', 'bg') |
| sig.threshold | numeric; what P value significance level to use (default = 0.05) |

## Value

a plot of type ggplot

## See Also

calculateBsFoldChange

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])

# make testset
bds = makeBindingSites(bds, bsSize = 7)
bds = assignToGenes(bds, anno.genes = gns)
bds = imputeBsDifferencesForTestdata(bds)
bds = calculateBsBackground(bds, anno.genes = gns, use.offset = FALSE)

# use all filters and remove binding sites that fail (default settings)
bds = filterBsBackground(bds)

# calculate fold-changes
bds = calculateBsFoldChange(bds)

# make volcano plot
plotBsVolcano(bds)
```

---

# processingStepsFlowChart

*Step-wise flowchart plot*

---

## Description

An overview plot that shows all workflow functions that were executed on the current object, with all input and output binding site numbers and major options that were used. The function can be called at any time in the analysis. Most optimal usage is after a full run of the wrapper function BSFind.

## Usage

```
processingStepsFlowChart(object, size.all = 3)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| size.all | numeric; size of all numbers |

## Value

a plot of type ggplot

## See Also

BSFind

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
bds = BSFind(bds, anno.genes = gns, anno.transcriptRegionList = regions,
 est.subsetChromosome = "chr22")
processingStepsFlowChart(bds)
```

---

# processingStepsTable *Create a table of all workflow steps for reporting*

---

## Description

Function that creates a printable table with all steps and numbers for each of the workflow steps that were carried out.

## Usage

```
processingStepsTable(object, option = c("reduced", "full", "extended"))
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored ranges |
| option | character; how detailed the table should be |

## Details

If option is set to 'reduced', only the most necessary information are collected. Option 'full' contains a full list of all options and parameters that were set in any of the workflow functions. Option 'extended' contains extra information about the binding site merging step.

## Value

a kableExtra table

## See Also

BSFind

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# apply 5% filter
bds = pureClipGlobalFilter(object = bds, cutoff = 0.05)
processingStepsTable(bds)
```

---

`pureClipGeneWiseFilter` *Filter PureCLIP sites by their score distribution per gene*

---

## Description

Function that applies a filter on the crosslink site score distribution at gene level. This allows to filter for those sites with the strongest signal on each gene. Since scores are tied to the expression level of the hosting transcript this function allows a fair filter for all genes partially independent of the expression level.

## Usage

```
pureClipGeneWiseFilter(
  object,
  cutoff = 0.05,
  overlaps = c("keepSingle", "removeAll", "keepAll"),
  anno.annoDB = NULL,
  anno.genes = NULL,
  match.score = "score",
  match.geneID = "gene_id",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object with stored crosslink ranges of width=1 |
| `cutoff` | numeric; defines the cutoff for which sites to remove, the smallest step is 1% (0.01). A cutoff of 5% will remove the lowest 5% sites, given their score, on each gene, thus keeping the strongest 95%. |
| `overlaps` | character; how overlapping gene loci should be handled. |
| `anno.annoDB` | an object of class `OrganismDbi` that contains the gene annotation (!!! Experimental !!!). |
| `anno.genes` | an object of class `GenomicRanges` that represents the gene ranges directly |
| `match.score` | character; meta column name of the crosslink site `GenomicRanges` object that holds the score which is used for sub-setting |

| | |
|---|---|
| match.geneID | character; meta column name of the genes GenomicRanges object that holds a unique geneID |
| quiet | logical; whether to print messages |

## Details

The GenomicRanges contained in the BSFDataSet need to have a meta-column that holds a numeric score value, which is used for filtering. The name of the column can be set with scoreCol.

In the case of overlapping gene annotation, a single crosslink site will be attributed to multiple genes. The overlaps parameter allows to control these cases. Option 'keepSingle' will only keep a single instance of the site; 'removeAll' will remove both sites; 'keepAll' will keep both sites.

The function is part of the standard workflow performed by BSFind.

## Value

an object of class BSFDataSet with its ranges filtered by those that passed the gene-wise threshold set with cutoff

## See Also

BSFind, estimateBsWidthPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# Load GRanges with genes
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
# apply 5% gene-wise filter
pureClipGeneWiseFilter(object = bds, anno.genes = gns, cutoff = 0.5, overlaps = "keepSingle")
```

---

# pureClipGlobalFilter *Filter PureCLIP sites by their score distribution*

---

## Description

Function that applies a filter on the global crosslink site score distribution. The GenomicRanges contained in the BSFDataSet need to have a meta-column that holds a numeric score value, which is used for filtering. The name of the column can be set with match.score.

## Usage

```
pureClipGlobalFilter(
  object,
  cutoff = 0.01,
  match.score = "score",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object with stored crosslink ranges of width=1 |
| cutoff | numeric; defines the cutoff for which sites to keep, the smallest step is 1% (0.01) |
| match.score | character; meta column name of the crosslink site GenomicRanges object that holds the score which is used for sub-setting |
| quiet | logical; whether to print messages |

## Details

The function is part of the standard workflow performed by BSFind.

## Value

an object of class BSFDataSet with its ranges filtered by those that passed the threshold set with cutoff

## See Also

BSFind, pureClipGlobalFilterPlot

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# apply 5% filter
pureClipGlobalFilter(object = bds, cutoff = 0.05)
```

---

# pureClipGlobalFilterPlot

*Plot the PureCLIP score distribution with global cutoff indicator*

---

## Description

A diagnostic function that plots the PureCLIP score distribution on a log2 scale. The function pureClipGlobalFilter is expected to be executed prior to calling this plot function.

## Usage

```
pureClipGlobalFilterPlot(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |

## Value

a plot of type ggplot

## See Also

pureClipGlobalFilter

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
# apply 5% filter
bds = pureClipGlobalFilter(object = bds, cutoff = 0.05)
pureClipGlobalFilterPlot(bds)
```

---

# quickFigure *Quick figures*

---

## Description

Summarize all results in a set of quick figures. Depending on how the function is called a different set of analytic plots are arranged into either a 'main' or 'supplementary' type multi-panel figure.

## Usage

```
quickFigure(
  object,
  what = c("main", "supp"),
  save.filename = NULL,
  save.width = 10,
  save.height = 12,
  save.device = "pdf",
  quiet = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| what | character; the plotting option. One of: 'main', 'supp' |
| save.filename | File name to create on the disc |
| save.width | numeric; plot size width |
| save.height | numeric; plot size height |
| save.device | charcter; Device to use. One of: 'pdf', 'png', ... |
| quiet | whether to print messages |
| ... | further arguments passed to ggsave |

### Value

a plot

### See Also

BSFind

### Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
bds = BSFind(bds, anno.genes = gns, anno.transcriptRegionList = regions,
 est.subsetChromosome = "chr22")
quickFigure(bds)
```

---

## rangeCoveragePlot *Plot crosslink event coverage over binding site range*

---

### Description

A diagnostic plot function that allows to check the coverage of crosslink events over different merged regions. The coverage is shown as mean over all replicates and conditions, with a standard deviation corridor.

### Usage

```
rangeCoveragePlot(
  object,
  width = 20,
  show.samples = FALSE,
  subset.chromosome = "chr1",
  quiet = TRUE
)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet, or a list of BSFDataSet |
| width | numeric; set the plotting range to show (in nt) |
| show.samples | logical; to show individual samples as lines |
| subset.chromosome | character; subset by a all ranges on the indicated chromosome. Can also be a vector with multiple chromosomes. If NULL then all ranges are being used. |
| quiet | logical; whether to print messages |

## Details

If object is a single BSFDataObject a single coverage plot will be drawn, whereas if it is a list of BSFDataObjects, then faceting is used to make a plot for each list element.

## Value

a plot of type ggplot2 displaying the crosslink coverage over the ranges of the given BSFDataSet

## See Also

BSFDataSet, makeBindingSites

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# plotting a single object
bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1)
rangeCoveragePlot(bds, subset.chromosome = "chr22")

# plotting multiple objects
bds1 <- makeBindingSites(object = bds, bsSize = 3, minWidth = 2,
minCrosslinks = 2, minClSites = 1, sub.chr = "chr22")
bds2 <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1, sub.chr = "chr22")
l = list(`1. bsSize = 3` = bds1, `2. bsSize = 9` = bds2)
rangeCoveragePlot(l, subset.chromosome = "chr22")
```

`reproducibilityCutoffPlot` *Plot to that shows how many replicates support each binding site*

## Description

Plotting function for settings specified in `reproducibilityFilter`.

## Usage

```
reproducibilityCutoffPlot(
  object,
  cutoff = 0.05,
  min.crosslinks = 1,
  max.range = 20,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `cutoff` | a vector of length = 1, or of length = levels(meta$conditions) with a single number (between 0-1) indicating the quantile cutoff |
| `min.crosslinks` | numeric of length = 1, defines the lower boundary for the minimum number of crosslinks a binding site has to be supported by all replicates, regardless of the replicate specific quantile threshold |
| `max.range` | maximum number of crosslinks per sites that should be shown |
| `...` | further arguments passed to ggplot |

## Value

a plot of type `ggplot2` showing the per replicate reproducibility cutoffs based on a given quantile threshold

## See Also

`reproducibilityFilter`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# merge binding sites
bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 1)
```

```
# use the same cutoff for both conditions
suppressWarnings(reproducibilityCutoffPlot(bds, max.range = 20, cutoff = c(0.05)))

# use different cutoffs for each condition
suppressWarnings(reproducibilityCutoffPlot(bds, max.range = 20, cutoff = c(0.1)))
```

---

`reproducibilityFilter` *Replicate reproducibility filter function*

---

## Description

For each replicate the number of binding sites with a certain number of crosslinks is calculated. A quantile based threshold (`cutoff`) is applied to each replicate. This indicates how many of the merged binding sites are supported by crosslinks from the respective replicate. Next, one can specify how many replicates need to pass the defined threshold for a binding site to be considered reproducible.

## Usage

```
reproducibilityFilter(
  object,
  cutoff = NULL,
  nReps = NULL,
  minCrosslinks = 1,
  returnType = c("BSFDataSet", "data.frame"),
  n.reps = lifecycle::deprecated(),
  min.crosslinks = lifecycle::deprecated(),
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `cutoff` | numeric; percentage cutoff to be used for the reproducibility quantile filtering |
| `nReps` | numeric; number of replicates that must meet the cutoff defined in `cutoff` for a binding site to be called reproducible. Defaults to N-1. |
| `minCrosslinks` | numeric; minimal number of crosslinks a binding site needs to have to be called reproducible. Acts as a lower boundary for `cutoff`. Defaults to 1. |
| `returnType` | one of "BSFDataSet" or "data.frame". "BSFDataSet" is the default and "matrix" can be used for easy plotting. |
| `n.reps` | deprecated -> use nReps instead |
| `min.crosslinks` | deprecated -> use minCrosslinks instead |
| `quiet` | logical; whether to print messages |

### Details

If `cutoff` is a single number then the indicated cutoff will be applied to all replicates. If it is a vector then each element in the vector is applied to all replicates of the respective condition. The order is hereby given by the levels of the condition column of the meta data (see `BSFDataSet`,`getMeta`). If the condition specific filter is applied, a meta column is added to the GRanges of the `BSFDataSet` object, indicating the support for each condition.

If `nReps` is a single number then this number is used as treshold for all binding sites. If it is a vector then it is applied to the replicates of the respective condition (like in `cutoff`). This allows the application of different thresholds for experiments of different experimental conditions. If the condition specific filter is applied, a meta column is added to the GRanges of the `BSFDataSet` object, indicating the support for each condition.

The function is part of the standard workflow performed by `BSFind`.

### Value

an object of type BSFDataSet

### See Also

`BSFind`, `reproducibilityFilterPlot`, `reproducibilitySamplesPlot`, `reproducibilityScatterPlot`

### Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

# merge binding sites
bds <- makeBindingSites(object = bds, bsSize = 9)

# use default return with condition specific threshold
bds = reproducibilityFilter(bds, cutoff = 0.1, nReps = 1)
```

---

## reproducibilityFilterPlot

*Plot to that shows the crosslink site distribution per replicate*

---

### Description

A diagnostic function that plots the number of crosslinks sites over the number of crosslink in these sites and highlights the replicate specific reproducibility cutoff that is derived from that distribution. The function `reproducibilityFilter` is expected to be executed prior to calling this plot function.

### Usage

```
reproducibilityFilterPlot(object, plotRange = 20)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| plotRange | numeric; number of crosslinks per sites that should be shown before summing them up |

### Value

a plot of type ggplot

### See Also

reproducibilityFilter

### Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds = makeBindingSites(object = bds, bsSize = 9)
bds = reproducibilityFilter(bds)
reproducibilityFilterPlot(bds)
```

---

## reproducibilitySamplesPlot

*UpSet-plot to that shows how each replicate supports binding sites*

---

### Description

A diagnostic function that plots the set sizes for each replicate, indicating how many binding site the specific replicate supports given its specific threshold. The function reproducibilityFilter is expected to be executed prior to calling this plot function.

### Usage

```
reproducibilitySamplesPlot(
  object,
  nIntersections = 20,
  show.title = TRUE,
  text.size = NULL
)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| nIntersections | numeric; number of intersection to be shown |
| show.title | logical; if plot title should be visible |
| text.size | numeric; fontsize of all numbers on axis |

## Value

a plot of type ggplot

## See Also

reproducibilityFilter, reproducibilityFilterPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds = makeBindingSites(object = bds, bsSize = 9)
bds = reproducibilityFilter(bds)
reproducibilitySamplesPlot(bds)
```

---

# reproducibilityScatterPlot

*Plot that shows binding site reproducibility as scatter*

---

## Description

Function compute the number of crosslinks per binding site on a log2 scale for each sample. Samples are pairwise correlated as a scatter and pairwise pearson correlation is shown.

## Usage

```
reproducibilityScatterPlot(object, quiet = FALSE)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| quiet | logical; whether to print messages |

## Details

Unlike most plotting functions, this function is actively calculating the values to plot.

## Value

an object of class ggplot2

## See Also

BSFind, reproducibilityFilter

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
bds <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
minCrosslinks = 2, minClSites = 3, sub.chr = "chr22")
reproducibilityScatterPlot(bds)
```

---

`setMeta` *Setter method for the meta data of the BSFDataSet object*

---

## Description

Meta data is stored as a `data.frame` and must contain the columns "condition", "clPlus" and "clMinus".

## Usage

```
setMeta(object, ...)

## S4 method for signature 'BSFDataSet'
setMeta(object, newMeta)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `...` | additional arguments |
| `newMeta` | the replacement meta data table |

## Value

an object of type `BSFDataSet` with updated meta data

## See Also

`BSFDataSet`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

nMeta = getMeta(bds)
setMeta(bds, nMeta)
```

setName *Setter method for the names of the BSFDataSet object The name slot holds the name information of the dataset*

## Description

Setter method for the names of the BSFDataSet object The name slot holds the name information of the dataset

## Usage

```
setName(object, ...)

## S4 method for signature 'BSFDataSet'
setName(object, newName)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| ... | additional arguments |
| newName | a character that is the name |

## Value

object of type BSFDataSet with updated name

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bdsNew = setName(bds, "test01")
```

setRanges *Setter method for the ranges of the BSFDataSet object The GRanges object that holds the genomic ranges information can be replaced.*

## Description

Setter method for the ranges of the BSFDataSet object The GRanges object that holds the genomic ranges information can be replaced.

## Usage

```
setRanges(object, ...)

## S4 method for signature 'BSFDataSet'
setRanges(object, newRanges, dropSeqlevels = TRUE, quiet = FALSE)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| ... | additional arguments |
| newRanges | an object of type GRanges |
| dropSeqlevels | enforce seqnames to be the same in ranges and signal, by dropping unused seqlevels which is required for most downstream functions such as coverageOverRanges |
| quiet | logical; whether to print messages |

## Value

object of type BSFDataSet with updated ranges

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

rng = getRanges(bds)
rng = rng + 10
bdsNew = setRanges(bds, rng)
```

setSignal *Setter method for the signal data of the BSFDataSet object*

## Description

Signal data is loaded from the path specified in `getMeta` columns "clPlus" and "clMinus" and stored as a list of RLE lists.

## Usage

```
setSignal(object, ...)

## S4 method for signature 'BSFDataSet'
setSignal(object, newSignal, dropSeqlevels = TRUE, quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `...` | additional arguments |
| `newSignal` | list of RLE lists |
| `dropSeqlevels` | enforce seqnames to be the same in ranges and signal, by dropping unused seqlevels which is required for most downstream functions such as `coverageOverRanges` |
| `quiet` | logical; whether to print messages |

## Value

an object of type `BSFDataSet` with updated signal

## See Also

`BSFDataSet`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

sgn = getSignal(bds)
sgn = lapply(sgn, function(selStrand){
   lapply(selStrand, function(chrList){
       chrList[names(chrList) == "chr22"]
   })
})
bdsNew = setSignal(bds, sgn)
```

setSummary *Setter method for the summary slot of the BSFDataSet object*

## Description

The summary slot is used to track information of the filtering steps applied in the makeBindingSites function

## Usage

```
setSummary(object, ...)

## S4 method for signature 'BSFDataSet'
setSummary(object, summary)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| ... | additional arguments |
| summary | a data.frame with the summary information to be stored in BSFDataSet |

## Value

an object of type BSFDataSet with updated summary info

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

df = data.frame(processingStep = c(1,2),
parameter = c(3,4))
bds = setSummary(bds, df)
```

show *Show method to for the BSFDataSet*

## Description

Prints the information for each of the input data slots in the BSFDataSet object.

## Usage

```
## S4 method for signature 'BSFDataSet'
show(object)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |

## Value

shows the current object state

## See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

show(bds)
```

subset-BSFDataSet *Subset a BSFDataSet object*

## Description

You can subset BSFDataSet by identifier or by position using the `[` operator. Empty seqlevels are being droppend after the subset.

## Usage

```
## S4 method for signature 'BSFDataSet,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]
```

### Arguments

| | |
|---|---|
| x | A BSFDataSet object. |
| i | Position of the identifier or the name of the identifier itself. |
| j | Not used. |
| ... | Additional arguments not used here. |
| drop | if the signal not covered by the subsetted ranges should be dropped or not |

### Value

A BSFDataSet object.

### Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

bdsNew = bds[1:10]
```

---

## summary *Summary method to for the BSFDataSet*

---

### Description

Prints the summaryinformation for the BSFDataSet object. This includes information on samples, conditions and crosslinks.

### Usage

```
## S4 method for signature 'BSFDataSet'
summary(object)
```

### Arguments

| | |
|---|---|
| object | a BSFDataSet object |

### Value

summary of the current object

### See Also

BSFDataSet

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

summary(bds)
```

---

supportRatio *Support ratio function for BSFDataSet objects*

---

## Description

Functions that computes a ratio to determine how well a given binding site with is supported by the crosslink coverage of the data. For a given `BSFDataSet` object binding sites are computed for each width indicated in the `bsWidths` vector (using the `coverageOverRanges` function). These coverages are compared to the coverage of regions flanking the binding sites. If not indicated in `bsFlank` these regions are of the same width as the binding sites.

## Usage

```
supportRatio(object, bsWidths, bsFlank = NA, sub.chr = NA, ...)
```

## Arguments

| | |
|---|---|
| `object` | a BSFDataSet object |
| `bsWidths` | a numeric vector indicating the different binding site width to compute the ratio for |
| `bsFlank` | optional; a numeric vector of the same length as `bsWidth` used to specify the width of the flanking regions |
| `sub.chr` | chromosome identifier (eg, chr1, chr2) used for subsetting the BSFDataSet object. |
| `...` | further arguments passed to `makeBindingSites` |

## Details

Testing the width of 3nt for example, would result in a coverage within all 3nt wide binding sites (c1) and a coverage computed on the adjacent 3nt flanking the binding sites up- and downstream (f1, f2). Based on these numbers the ratio is computed by: c1/(1/2(f1+f2)).

The median over all ratios is reported as representative value.

## Value

an object of class `data.frame`

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

suppressWarnings(supportRatio(bds, bsWidths = c(3,7)))
```

---

supportRatioPlot *Plot that shows the binding site support ratio*

---

## Description

Function that shows a ratio to determine how well a given binding site with is supported by the crosslink coverage of the data. Ratios are computed using the supportRatio function.

## Usage

```
supportRatioPlot(object, bsWidths, bsFlank = NA, ...)
```

## Arguments

| | |
|---|---|
| object | a BSFDataSet object |
| bsWidths | a numeric vector indicating the different binding site width to compute the ratio for |
| bsFlank | optional; a numeric vector of the same length as bsWidth used to specify the width of the flanking regions |
| ... | further arguments passed to makeBindingSites |

## Details

The higher the ratio, the more does the given binding site width captures the enrichment of crosslinks compared the the local surrounding. A ratio equal to 1 would mean no enrichemnt at all.

## Value

an object of class ggplot2

## Examples

```
# load data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))

suppressWarnings(supportRatioPlot(bds, bsWidths = c(3,7),
minWidth = 1, minClSites = 1, minCrosslinks = 2))
```

`targetGeneSpectrumPlot` *Bar-chart to show the hosting gene types of binding sites*

## Description

A diagnostic function that plots the gene type of the hosting gene for each binding site. The function `assignToGenes` is expected to be executed prior to calling this plot function.

## Usage

```
targetGeneSpectrumPlot(object, showNGroups = 5, text.size = 4)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object |
| `showNGroups` | numeric; the number of different gene types to show |
| `text.size` | numeric; the size of the text elments on the plot |

## Value

a plot of type `ggplot`

## See Also

`assignToGenes` `geneOverlapsPlot`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
bds = makeBindingSites(object = bds, bsSize = 9)
bds = assignToGenes(bds, anno.genes = gns)
targetGeneSpectrumPlot(bds)
```

`transcriptRegionOverlapsPlot` *UpSet-plot to that shows the transcript region overlaps*

## Description

A diagnostic function that plots the transcript regions of binding sites on overlapping loci. The function `assignToTranscriptRegions` is expected to be executed prior to calling this plot function.

## Usage

```
transcriptRegionOverlapsPlot(object, text.size = NULL, show.title = TRUE)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object |
| `text.size` | numeric; fontsize of all numbers on axis |
| `show.title` | logical; if plot title should be visible |

## Value

a plot of type `ggplot`

## See Also

`assignToTranscriptRegions` `transcriptRegionSpectrumPlot`

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
bds = makeBindingSites(object = bds, bsSize = 9)
bds = assignToGenes(bds, anno.genes = gns)
bds = assignToTranscriptRegions(object = bds, anno.transcriptRegionList = regions)
transcriptRegionOverlapsPlot(bds)
```

`transcriptRegionSpectrumPlot` *Bar-chart to show the hosting transcript regions of binding sites*

## Description

A diagnostic function that plots the transcript regions of the hosting gene for each binding site. The function `assignToTranscriptRegions` is expected to be executed prior to calling this plot function.

## Usage

```
transcriptRegionSpectrumPlot(
  object,
  values = c("asis", "percentage"),
  normalize = FALSE,
  normalize.factor = c("sum", "median", "mean"),
  show.others = FALSE,
  text.size = 4
)
```

## Arguments

| | |
|---|---|
| `object` | a `BSFDataSet` object |
| `values` | character; if values should be presented 'as-is', that means for example as frequencies in case `normalization = FALSE`, or as percentages |
| `normalize` | logical; whether to normalize values |
| `normalize.factor` | character; indicate by what factor values should be normalized to region length by |
| `show.others` | logical; whether to show 'others' category. Has to be false if `normalize = TRUE` |
| `text.size` | numeric; font size of the numbers to be displayed on each bar |

## Details

Count frequencies can be normalized to the length of the hosting region with option `normalize`. The specific factor how the hosting region length is used is given by `normalize.factor`. In the case of `normalize.factor = "sum"` binding site frequencies are divided by the summed length of all regions that host the specific binding site.

Further with option `values` once can indicate whether raw or normalized frequencies should be shown 'as-is' or normalized to 'percentages'.

## Value

a plot of type `ggplot`

## See Also

assignToTranscriptRegions transcriptRegionOverlapsPlot

## Examples

```
# load clip data
files <- system.file("extdata", package="BindingSiteFinder")
load(list.files(files, pattern = ".rda$", full.names = TRUE))
load(list.files(files, pattern = ".rds$", full.names = TRUE)[1])
load(list.files(files, pattern = ".rds$", full.names = TRUE)[2])
bds = makeBindingSites(object = bds, bsSize = 9)
bds = assignToGenes(bds, anno.genes = gns)
bds = assignToTranscriptRegions(object = bds, anno.transcriptRegionList = regions)
transcriptRegionSpectrumPlot(bds)
```

# Index