Package 'gUtils'

November 6, 2017

Title R Package Providing Additional Capabilities and Speed for GenomicRanges Operations

Version 0.2.0

Description R package providing additional capabilities and speed for GenomicRanges operations.

Depends R (>= 3.1.0),
GenomicRanges (>= 1.18),
data.table (>= 1.9)

Imports IRanges (>= 2.0),
S4Vectors (>= 0.4),
GenomeInfoDb (>= 1.2),
parallel,
BiocGenerics(>= 0.12),
methods,
Matrix,
stringr

Suggests BSgenome.Hsapiens.UCSC.hg19,
testthat,
covr,
rtracklayer

License GPL-2

BugReports http://github.com/mskilab/gUtils/issues

LazyData true

RoxygenNote 6.0.1.9000

R topics documented:

`anchorlift` *anchorlift*

Description

"lifts" all queries with respect to subject in coordinates that are within "pad" i.e. puts the queries into subject-centric coordinates, which is a new genome with label "Anchor" (default)

Respects strand of subject (i.e. if subject strand gr is "-" then will lift all queries to the left of it into positive subject-centric coordinates). Keeps track of subject and query id for later deconvolution if need be.

Usage

```
anchorlift(query, subject, window = 1e+09, by = NULL, seqname = "Anchor",
  include.values = TRUE)
```

Arguments

`query`	GRanges that will be lifted around the subject
`subject`	GRanges around which the queries will be lifted
`window`	non-negative integer scalar specifying how far around each subject to gather query intervals to lift (default 1e9)
`by`	character vector specifying additional columsn (e.g. sample id) around which to restrict overlaps (via gr.findoverlaps) (default NULL)
`seqname`	Character specifying the name of the output sequence around which to anchor (default "Anchor")
`include.values`	logical flag whether to include values from query and subject (default TRUE)

Author(s)

Marcin Imielinski

dt2gr *Convert data.table to GRanges*

Description

Takes as input a data.table which must have the following fields: `start`, `end`, `strand`, `seqnames`. Will throw an error if any one of these is not present. All of the remaining fields are added as metadata to the `GRanges`.

Usage

```
dt2gr(dt, key = NULL, seqlengths = hg_seqlengths(), seqinfo = Seqinfo())
```

Arguments

dt	`data.table` to convert to `GRanges`
dt	data.table to convert

Value

`GRanges` object of `length = nrow(dt)`

Examples

```
gr <- dt2gr(data.table(start=c(1,2), seqnames=c("X", "1"), end=c(10,20), strand = c('+', '-')))
```

example_dnase *DNAaseI hypersensitivity sites for hg19A*

Description

DNAaseI hypersensitivity sites from UCSC Table Browser hg19, subsampled to 10,000 sites

Format

`GRanges`

example_genes *RefSeq genes for hg19*

Description

RefSeq genes with exon count and name

Format

`GRanges`

gr.breaks *Break GRanges at given breakpoints into disjoint gr*

Description

Break GRanges at given breakpoints into disjoint gr

Usage

```
gr.breaks(bps = NULL, query = NULL)
```

Arguments

bps	GRanges of width 1, locations of the bp; if any element width larger than 1, both boundary will be considered individual breakpoints
query	a disjoint GRanges object to be broken

Value

GRanges disjoint object at least the same length as query, with a metadata column qid indicating input index where new segment is from

Author(s)

Xiaotong Yao

gr.chr *Prepend "chr" to* GRanges seqlevels

Description

Prepend "chr" to GRanges seqlevels

Usage

```
gr.chr(gr)
```

Arguments

gr	GRanges object to append 'chr' to

Value

Identical GRanges, but with 'chr' prepended to each seqlevel

Examples

```
gr <-  gr.chr(GRanges(c(1,"chrX"), IRanges(c(1,2), 1)))
seqnames(gr)
```

gr.collapse *Collapse adjacent ranges*

Description

Like GenomicRanges::reduce except only collapses «adjacent» ranges in the input

Usage

```
gr.collapse(gr, pad = 1)
```

Arguments

gr	GRanges to collapse
pad	Padding that allows for not quite adjacent elements to be considered overlapping. 1

Value

Collapsed ranges

Author(s)

Marcin Imielinski

gr.cov *gr.cov*

Description

Sums granges either by doing coverage and either weighting them equally or using a field "weight". Will return either sum or average.

Most basic functionality is like an as(coverage(gr), 'GRanges')

Usage

```
gr.sum(gr, field = NULL, mean = FALSE)
```

Arguments

gr	GRanges to sum
field	metadata field from gr to use as a weight
mean	logical scalar specifying whether to divide the output at each interval but the total number of intervals overlapping it (only applies if field == NULL) (default FALSE)

Value

non-overlapping granges spanning the seqlengths of gr with $score (if field is NULL) or $field specifying the sum / mean at that position

gr.dice — *Dice up* GRanges *into* width = 1 GRanges *spanning the input (warning can produce a very large object)*

Description

Dice up GRanges into width = 1 GRanges spanning the input (warning can produce a very large object)

Usage

```
gr.dice(gr)
```

Arguments

gr	GRanges object to dice

Value

GRangesList where kth element is a diced pile of GRanges from kth input GRanges

Author(s)

Marcin Imielinski

Examples

```
gr.dice(GRanges(c(1,4), IRanges(c(10,10),20)))
```

gr.disjoin — *GenomicRanges disjoin with some spice Identical to GRanges disjoin, except outputs inherit metadata from first overlapping parent instance on input*

Usage

```
gr.disjoin(x, ..., ignore.strand = TRUE)
```

Arguments

x	GRanges to disjoin
...	arguments to disjoin
ignore.strand	logical scalar, default TRUE

gr.dist — *Pairwise distance between two* GRanges

Description

Computes matrix of pairwise distance between elements of two GRanges objects of length n and m.

Usage

```
gr.dist(gr1, gr2 = NULL, ignore.strand = FALSE, ...)
```

Arguments

gr1	First GRanges
gr2	Second GRanges
ignore.strand	Don't required elements be on same strand to avoid NA [FALSE]
...	Additional arguments to be supplied to GenomicRanges::distance

Details

Distances are computed as follows:

- NA for ranges on different seqnames
- 0 for overlapping ranges
- min(abs(end1-end2), abs(end1-start2), abs(start1-end2), abs(start1-end1),) for all others

If only gr1 is provided, then will return n x n matrix of gr1 vs itself
If max.dist = TRUE, then will replace min with max above

Value

N by M matrix with the pairwise distances, with gr1 on rows and gr2 on cols

Author(s)

Marcin Imielinski

gr.duplicated — *Allows to restrict duplicates using "by" columns and allows in exact matching*

Description

Allows to restrict duplicates using "by" columns and allows in exact matching

Usage

```
gr.duplicated(query, by = NULL, type = "any")
```

Arguments

query	query ranges

Examples

```
gr.duplicated(GRanges(c(1,1,1), IRanges(c(2,5,5), width=1)))

gr.duplicated(GRanges(c(1,1,1), IRanges(c(2,5,5), width=1)))
```

---

gr.end *Get the right ends of a* GRanges

Description

Alternative to GenomicRanges::flank that will provide end positions *within* intervals

Usage

```
gr.end(x, width = 1, force = FALSE, ignore.strand = TRUE, clip = TRUE)
```

Arguments

x	GRanges object to operate on
width	Specify subranges of greater width including the start of the range. [1]
force	Allows returned GRanges to have ranges outside of its Seqinfo bounds. [FALSE]
ignore.strand	If set to FALSE, will extend '-' strands from the other direction. [TRUE]
clip	Trims returned GRanges so that it does not extend beyond bounds of the input GRanges. [TRUE]

Value

GRanges object of width = width ranges representing end of each genomic range in the input.

Author(s)

Marcin Imielinski

Examples

```
gr.end(example_dnase, width=200, clip=TRUE)
```

`gr.findoverlaps` *Wrapper to* `GenomicRanges::findOverlaps` *with added functionality*

Description

Returns `GRanges` of matches with two additional fields:

`$query.id` - index of matching query `$subject.id` - index of matching subject

Optional `"by"` field is a character scalar that specifies a metadata column present in both query and subject that will be used to additionally restrict matches, i.e. to pairs of ranges that overlap and also have the same values of their `"by"` fields

Usage

```
gr.findoverlaps(query, subject, ignore.strand = TRUE, first = FALSE,
  qcol = NULL, scol = NULL, type = "any", by = NULL,
  return.type = "same", max.chunk = 1e+13, verbose = FALSE,
  mc.cores = 1, ...)
```

Arguments

`query`	Query `GRanges` pile
`subject`	Subject `GRanges` pile
`ignore.strand`	Don't consider strand information during overlaps. `[TRUE]`
`first`	Restrict to only the first match of the subject (default is to return all matches). `[FALSE]`
`qcol`	`character` vector of query meta-data columns to add to results
`scol`	`character` vector of subject meta-data columns to add to results
`type`	`type` argument as defined by `IRanges::findOverlaps` (`"any"`, `"start"`, `"end"`, `"within"`, `"equal"`). `["any"]`
`by`	Meta-data column to consider when performing overlaps [NULL]
`return.type`	Select data format to return (supplied as character): `"same"`, `"data.table"`, `"GRanges"`. `["same"]`
`max.chunk`	Maximum number of `query*subject` ranges to consider at once. Lower number increases runtime but decreased memory. If `length(query)*length(subject)` is less than `max.chunk`, overlaps will run in one batch.`[1e13]`
`verbose`	Increase the verbosity. `[FALSE]`
`mc.cores`	Number of cores to use when running in chunked mode
`...`	Additional arguments sent to `IRanges::findOverlaps`.

Value

`GRanges` pile of the intersection regions, with `query.id` and `subject.id` marking sources

gr.fix *"Fixes"* seqlengths / seqlevels

Description

If "genome" not specified will replace NA seqlengths in GRanges to reflect largest coordinate per seqlevel and removes all NA seqlevels after this fix.

Usage

```
gr.fix(gr, genome = NULL, gname = NULL, drop = FALSE)
```

Arguments

gr	GRanges object to fix
genome	Genome to fix to: Seqinfo, BSgenome, GRanges (w/seqlengths), GRangesList (w/seqlengths)
gname	Name of the genome (optional, just appends to Seqinfo of the output) [NULL]
drop	Remove ranges that are not present in the supplied genome [FALSE]

Details

if "genome" defined (i.e. as Seqinfo object, or a BSgenome, GRanges, GRangesList object with populated seqlengths), then will replace seqlengths in gr with those for that genome

Value

GRanges pile with the fixed Seqinfo

gr.flatten *Lay ranges end-to-end onto a derivate "chromosome"*

Description

Takes pile of GRanges and returns into a data.frame with nrow = length(gr) with each representing the corresponding input range superimposed onto a single "flattened" chromosome, with ranges laid end-to-end

Usage

```
gr.flatten(gr, gap = 0)
```

Arguments

gr	GRanges to flatten
gap	Number of bases between ranges on the new chromosome [0]

Value

data.frame with start and end coordinates, and all of the original metadata

gr.flipstrand *Flip strand on* GRanges

Description

Flip strand on GRanges

Usage

```
gr.flipstrand(gr)
```

Arguments

gr	GRanges pile with strands to be flipped

Value

GRanges with flipped strands (+ to -, * to *, - to *)

Examples

```
gr.flipstrand(GRanges(1, IRanges(c(10,10,10),20), strand=c("+","*","-")))
```

gr.in *Versatile implementation of* GenomicRanges::over

Description

returns T / F vector if query range i is found in any range in subject

Usage

```
gr.in(query, subject, ...)
```

Arguments

query	GRanges
subject	GRanges
...	Argument to be sent to gr.findoverlaps (e.g. by)

`gr.match` *Alternative* `GenomicRanges::match` *that accepts additional* `gr.findoverlaps` *options*

Description

Wrapper to `GenomicRanges::match` (uses `gr.findoverlaps`). This allows users to match on additional by fields, or chunk into smaller pieces for lower memory.

Usage

```
gr.match(query, subject, max.slice = Inf, verbose = FALSE, ...)
```

Arguments

`query`	Query `GRanges` pile
`subject`	Subject `GRanges` pile
`max.slice`	max slice of query to match at a time
`verbose`	whether to give verbose output
`...`	Additional arguments to be passed along to `gr.findoverlaps`.

Value

Vector of length = `length(query)` with subject indices of *first* subject in query, or NA if none found. This behavior is different from `gr.findoverlaps`, which will return *all* indicies of subject in query (in the case of one query overlaps with multiple subject) ... = additional args for findOverlaps (IRanges version)

Author(s)

Marcin Imielinski

`gr.merge` *merge GRanges using coordinates as primary key*

Description

Uses gr.findoverlaps to enable internal and external joins of GRanges using syntax similar to "merge" where merging is done using coordinates +/- "by" fields

Uses gr.findoverlaps / findOverlaps for heavy lifting, but returns outputs with metadata populated as well as query and subject ids. For external joins, overlaps x with gaps(y) and gaps(x) with y.

Usage

```
gr.merge(query, subject, by = NULL, all = FALSE, all.query = all,
  all.subject = all, verbose = FALSE, ignore.strand = TRUE, ...)
```

Arguments

query	query ranges
subject	subject
by	additional metadata fields to join on
all	whether to include left and right joins
all.query	whether to do a left join
all.subject	whether to do a right join

---

gr.mid *Get the midpoints of* GRanges *ranges*

Description

Get the midpoints of GRanges ranges

Usage

```
gr.mid(x)
```

Arguments

x	GRanges object to operate on

Value

GRanges of the midpoint, calculated from `floor(width(x)/2)`

Examples

```
gr.mid(GRanges(1, IRanges(1000,2000), seqinfo=Seqinfo("1", 2000)))
```

---

gr.nochr *Remove chr prefix from GRanges seqlevels*

Description

Remove chr prefix from GRanges seqlevels

Usage

```
gr.nochr(gr)
```

Arguments

gr	GRanges with chr seqlevel prefixes

Value

GRanges without chr seqlevel prefixes

gr.pairflip *Create pairs of ranges and their strand-inverse*

Description

From a GRanges returns a GRangesList with each item consisting of the original GRanges and its strand flip

Usage

```
gr.pairflip(gr)
```

Arguments

gr	GRanges

Value

GRangesList with each element of length 2

gr.rand *Generate random* GRanges *on genome*

Description

Randomly generates non-overlapping GRanges with supplied widths on supplied genome. Seed can be supplied with set.seed

Usage

```
gr.rand(w, genome)
```

Arguments

w	Vector of widths (length of w determines length of output)
genome	Genome which can be a GRanges, GRangesList, or Seqinfo object. Default is "hg19" from the BSGenome package.

Value

GRanges with random intervals on the specifed "chromosomes"

Note

This function is currently quite slow, needs optimization

Author(s)

Marcin Imielinski

Examples

```
## Generate 5 non-overlapping regions of width 10 on hg19
gr.rand(rep(10,5), BSgenome.Hsapiens.UCSC.hg19::Hsapiens)
```

`gr.sample` *Randomly sample* `GRanges` *intervals within territory*

Description

Samples k intervals of length "len" from a pile of `GRanges`.

- If k is a scalar then will (uniformly) select k intervals from the summed territory of `GRanges`
- If k is a vector of length(gr) then will uniformly select k intervals from each.

Usage

```
gr.sample(gr, k, wid = 100, replace = TRUE)
```

Arguments

`gr`	`GRanges` object defining the territory to sample from
`k`	Number of ranges to sample
`wid`	Length of the `GRanges` element to produce [100]
`replace`	If TRUE, will bootstrap, otherwise will sample without replacement. [TRUE]

Value

GRanges of max length sum(k) [if k is vector) or k*length(gr) (if k is scalar) with labels indicating the originating range.

sample 5 `GRanges` of length 10 each from territory of RefSeq genes gr.sample(reduce(example_genes), k=5, wid=10)

Note

This is different from `GenomicRanges::sample` function, which just samples from a pile of `GRanges`

Author(s)

Marcin Imielinski

gr.simplify — *Simplify granges by collapsing all non-overlapping adjacent ranges that share a given "field" value (adjacent == adjacent in the input GRanges object)*

Description

Simplify granges by collapsing all non-overlapping adjacent ranges that share a given "field" value (adjacent == adjacent in the input GRanges object)

Usage

```
gr.simplify(gr, field = NULL, val = NULL, include.val = TRUE,
  split = FALSE, pad = 1)
```

Arguments

gr	takes in gr or grl
field	character scalar, corresponding to value field of gr. [NULL]
val	[NULL]
include.val	scalar logical, will include in out gr values field of first matching record in input gr. [TRUE]
split	Split the output into GRangesList split by "field". [FALSE]
pad	Pad ranges by this amount before doing merge. [1], which merges contiguous but non-overlapping ranges.

Value

Simplified GRanges with "field" populated with uniquely contiguous values

gr.start — *Get GRanges corresponding to beginning of range*

Description

Get GRanges corresponding to beginning of range

Usage

```
gr.start(x, width = 1, force = FALSE, ignore.strand = TRUE, clip = TRUE)
```

Arguments

x	GRanges object to operate on
width	[default = 1] Specify subranges of greater width including the start of the range.
force	[default = F] Allows returned GRanges to have ranges outside of its Seqinfo bounds.
ignore.strand	If set to FALSE, will extend '-' strands from the other direction [TRUE].
clip	[default = F] Trims returned GRanges so that it does not extend beyond bounds of the input GRanges

Value

GRanges object of width 1 ranges representing start of each genomic range in the input.

Examples

```
gr.start(example_dnase, width=200)
gr.start(example_dnase, width=200, clip=TRUE)
```

---

gr.string *Return UCSC style interval string corresponding to* GRanges *pile (ie chr:start-end)*

Description

Return UCSC style interval string corresponding to GRanges pile (ie chr:start-end)

Usage

```
gr.string(gr, add.chr = FALSE, mb = FALSE, round = 3, other.cols = c(),
  pretty = FALSE)
```

Arguments

gr	GRanges pile to get intervals from
add.chr	Prepend seqnames with "chr" [FALSE]
mb	Round to the nearest megabase [FALSE]
round	If mb supplied, how many digits to round to. [3]
other.cols	Names of additional mcols fields to add to the string (seperated by ";")

Author(s)

Marcin Imielinski

Examples

```
gr.string(example_genes, other.cols = c("name", "name2"))
```

gr.stripstrand *gr.stripstrand*

Description

Sets strand to "*"

Usage

```
gr.stripstrand(gr)
```

Arguments

gr	GRanges to remove strand information from

Value

GRanges with strand set to *

gr.sub *Apply* gsub *to seqlevels of a* GRanges

Description

Apply gsub to seqlevels of gr, by default removing 'chr', and "0.1" suffixes, and replacing "MT" with "M"

Usage

```
gr.sub(gr, a = c("(^chr)(\\.1$)", "MT"), b = c("", "M"))
```

Arguments

gr	GRanges to switch out seqlevels for
a	Vector of regular expressions of things to sub-out
b	Vector of values to sub in

gr.tile — *Tile ranges across* GRanges

Description

Tiles interval (or whole genome) with segments of <= specified width.

Usage

```
gr.tile(gr, w = 1000)
```

Arguments

gr	GRanges, seqlengths or Seqinfo range to tile. If has GRanges has overlaps, will reduce first.
w	Width of each tile

Examples

```
## 10 tiles of width 10
gr1 <- gr.tile(GRanges(1, IRanges(1,100)), w=10)
## make them overlap each other by 5
gr1 + 5
```

gr.tile.map — *gr.tile.map*

Description

Given two tilings of the genome (e.g. at different resolution) query and subject outputs a length(query) list whose items are integer vectors of indices in subject overlapping that overlap that query (strand non-specific)

Usage

```
gr.tile.map(query, subject, verbose = FALSE)
```

Arguments

query	Query GRanges pile, perhaps created from some tile (e.g. gr.tile), and assumed to have no gaps
subject	Subject GRanges pile, perhaps created from some tile (e.g. gr.tile), and assumed to have no gaps
verbose	Increase the verbosity of the output

Value

list of length = length(query), where each element i is a vector of indicies in subject that overlaps element i of query

Note

Assumes that input query and subject have no gaps (including at end) or overlaps, i.e. ignores end() coordinates and only uses "starts"

Author(s)

Marcin Imielinski

`gr.trim` *Trims pile of* `GRanges` *relative to the specified <local> coordinates of each range*

Description

Example: `GRanges` with genomic coordinates 1:1,000,000-1,001,000 can get the first 20 and last 50 bases trimmed off with `start = 20, end = 950`. if end is larger than the width of the corresponding gr, then the corresponding output will only have `end(gr)` as its coordinate.

Usage

```
gr.trim(gr, starts = 1, ends = 1)
```

Arguments

`gr`	`GRanges` to trim
`starts`	Number of bases to trim off of the front`[1]`
`ends`	Number of bases to trim off of the back`[1]`

Details

This is a role not currently provided by the standard `GenomicRanges` functions (e.g. `shift`, `reduce`, `restrict`, `shift`, `resize`, `flank`)

Examples

```
## trim the first 20 and last 50 bases
gr.trim(GRanges(1, IRanges(1e6, width=1000)), starts=20, ends=950)
## return value: GRanges on 1:1,000,019-1,000,949
```

gr.val *Annotate* GRanges *with values from another* GRanges

Description

Annotates GRanges in query with aggregated values of GRanges in target in field val. If val is numeric: given target with value column target representing ranged data (i.e. segment intensities), thn computes the value in each query GRanges as the weighted mean of its intersection with target (ie the target values weighted by the width of the intersections).

Usage

```
gr.val(query, target, val = NULL, mean = TRUE, weighted = mean,
  na.rm = FALSE, by = NULL, by.prefix = val, merge = FALSE,
  verbose = FALSE, FUN = NULL, default.val = NA, max.slice = Inf,
  mc.cores = 1, ..., sep = ", ")
```

Arguments

query	GRanges of query ranges whose val column we will populate with aggregated values of target
target	GRanges of target ranges that already have "val" column populated
val	If a character field: then aggregation will paste together the (unique), overlapping values, collapsing by comma. [NULL]
mean	Scalar logical flag. If FALSE then will return sum instead of mean, only applies if target val column is numeric.
weighted	Calculate a weighted mean. If FALSE, calculates unweighted mean. [TRUE]
na.rm	Remove NA values when calulating means. only applies if val column of target is numeric [FALSE]
by	scalar character, specifies additional "by" column of query AND target that will be used to match up query and target pairs (i.e. in addition to pure GRanges overlap), default is NULL
by.prefix	Choose a set of val fields by a shared prefix.
merge	if merge = FALSE then will cross every range in query with every level of "by" in target (and create data matrix), otherwise will assume query has "by" and merge only ranges that have matching "by" values in both query and target
verbose	Increase the verbosity of the output
FUN	Optional different function to call than mean. Takes two arguments (value, na.rm = TRUE) if weighted = FALSE, and three (value, width, na.rm = TRUE) if weighted = TRUE
default.val	If no hit in target found in query, fill output val field with this value.
max.slice	Maximum number of query ranges to consider in one memory chunk. [Inf]
mc.cores	Number of cores to use when running in chunked mode
...	Additional arguments to be sent to gr.findoverlaps.
sep	scalar character, specifies character to use as separator when aggregating character "vals" from target, only applies if target is character

Details

Applications include (among others):

- Querying the average value of target across a given query interval (e.g. exon to gene pileup)
- recasting a high res tiling in terms of low res intervals.

Usually query intervals are bigger than the target intervals.

Value

`query` with the `val` field populated

Note

`query` and `target` can be `GRangesList` object, in which case val will refer to `GRangesList` level values fields

Author(s)

Marcin Imielinski

`gr2dt` *Converts* `GRanges` *to* `data.table`

Description

and a field grl.iix which saves the (local) index that that gr was in its corresponding grl item

Usage

```
gr2dt(x)
```

Arguments

`x`	`GRanges` to convert

`grbind` *Concatenate* `GRanges`*, robust to different* `mcols`

Description

Concatenates `GRanges` objects, taking the union of their features if they have non-overlapping features

Usage

```
grbind(x, ...)
```

Arguments

x	First GRanges
...	additional GRanges

Value

Concatenated GRanges grbind(example_genes, example_dnase)

Note

Does not fill in the Seqinfo for the output GRanges

---

grl.eval — *evaluate and aggregate expression on GRanges column in GRangesList*

Description

Evaluate expression expr on indivdual granges inside grangeslist. Expression should result in a single i.e. scalar value per grangeslist item.

Usage

```
grl.eval(grl, expr, condition = NULL)
```

Arguments

grl	GRangesList to eval over
expr	expression on columns of granges or granges list
condition	optional expression (with logical or integer output) on columns of GRanges on which to subset prior to evaluating main expr

---

grl.hiC — *HiC data for chr14 from Lieberman-Aiden 2009 (in hg19), subsampled to 10,000 interactions*

Description

HiC data for chr14 from Lieberman-Aiden 2009 (in hg19), subsampled to 10,000 interactions

Format

GRangesList

grl.in *Check intersection of* GRangesList *with windows on genome*

Description

Like only if the ranges in grl[i] intersect «all», «some», «only» windows in the subject

Usage

```
grl.in(grl, windows, some = FALSE, only = FALSE, logical = TRUE,
  exact = FALSE, ignore.strand = TRUE, ...)
```

Arguments

grl	GRangesList object to query for membership in windows
windows	GRanges pile of windows
some	Will return TRUE for GRangesList elements that intersect at least on window range [FALSE]
only	Will return TRUE for GRangesList elements only if there are no elements of query that fail to interesect with windows [FALSE]
logical	will return logical otherwise will return numeric vector of number of windows overlapping each grl
...	Additional parameters to be passed on to GenomicRanges::findOverlaps

Details

eg can use to identify read pairs whose ends are contained inside two genes)

grl.pivot *Pivot a* GRangesList*, inverting "x" and "y"*

Description

"Pivots" grl object "x" by returning a new grl "y" whose kth item is gr object of ranges x[[i]][k] for all i in 1:length(x) e.g. If length(grl) is 50 and length of each GRanges element inside is 2, then grl.pivot will produce a length 3 GRangesList with 50 elements per GRanges

Usage

```
grl.pivot(x)
```

Arguments

x	GRangesList object to pivot

Details

Assumes all grs in "x" are of equal length

Examples

```
grl.pivot(grl.hiC)
```

`grl.reduce` *grl.reduce*

Description

Quickly ranges inside grl +/- pad Can use with split / unlist

Usage

```
grl.reduce(grl, pad = 0, clip = FALSE)
```

Arguments

grl	GRangesList
pad	padding to add to ranges inside grl before reduing

Value

`GRangesList` with reduced intervals

Author(s)

Marcin Imielinski

Examples

```
grl.reduce(grl, 1000)

unlist(grl.reduce(split(reads+10000, reads$BX)))
```

`grl.string` *Create string representation of* `GRangesList`

Description

Return ucsc style interval string corresponding to each `GRanges` in the `GRangesList`. One line per per `GRangesList` item. `GRanges` elements themselves are separated by `sep`

Usage

```
grl.string(grl, mb = FALSE, sep = ",", ...)
```

Arguments

grl	GRangesList to convert to string vector
mb	Will return as MB and round to "round" [FALSE]
sep	Character to separate single GRanges ranges [,]
...	Additional arguments to be passed to gr.string

Value

Character vector where each element is a GRanges pile corresponding to a single GRangesList element

Author(s)

Marcin Imielinski

Examples

```
grl.string(grl.hiC, mb=TRUE)
```

---

grl.stripnames *Remove* GRanges *names inside a* GRangesList

Description

Remove GRanges names inside a GRangesList

Usage

```
grl.stripnames(grl)
```

Arguments

grl	GRangesList with names elements

Value

GRangesList where GRanges have no names

grl.unlist *Robust unlisting of* GRangesList *that keeps track of origin*

Description

Does a "nice" unlist of a GRangesList object adding a field grl.ix denoting which element of the GRangesList each GRanges corresponds to and a field grl.iix which saves the (local) index that that gr was in its corresponding GRangesList item

Usage

```
grl.unlist(grl)
```

Arguments

grl	GRangeList object to unlist

Details

In this way, grl.unlist is reversible, while BiocGenerics::unlist is not.

Value

GRanges with added metadata fields grl.ix and grl.iix.

Examples

```
grl.unlist(grl.hiC)
```

grl1 *Fake rearrangement data (set 1)*

Description

Fake rearrangement data (set 1)

Format

GRangesList

grl2 *Fake rearrangement data (set 2)*

Description

Fake rearrangement data (set 2)

Format

GRangesList

grlbind *Concatenate* GRangesList *objects.*

Description

Concatenates GRangesList objects taking the union of their mcols features if they have non-overlapping features

Usage

```
grlbind(...)
```

Arguments

...	Any number of GRangesList to concatenate together

Value

Concatenated GRangesList with NA filled in for mcols fields that are non-overlapping. Note that the elements are re-named with sequential numbers

Author(s)

Marcin Imielinski

Examples

```
## Concatenate
grl.hiC2 <- grl.hiC[1:20]
mcols(grl.hiC2)$test = 1
grlbind(grl.hiC2, grl.hiC[1:30])
```

hg_seqlengths *Output standard human genome seqlengths*

Description

Outputs a standard seqlengths for human genome +/- "chr".

Usage

```
hg_seqlengths(genome = NULL, chr = FALSE, include.junk = FALSE)
```

Arguments

genome	A BSgenome or object with a seqlengths accessor. Default is hg19, but loads with warning unless explicitly provided
chr	Flag for whether to keep "chr". Default FALSE
include.junk	Flag for whether to not trim to only 1-22, X, Y, M. Default FALSE

Value

Named integer vector with elements corresponding to the genome seqlengths

Note

A default genome can be set with the environment variable DEFAULT_BSGENOME. This can be the full namespace of the genome e.g.: `DEFAULT_BSGENOME=BSgenome.Hsapiens.UCSC.hg19::Hsapiens` OR a URL / file path pointing to a chrom.sizes text file (e.g. http://genome.ucsc.edu/goldenpath/help/hg19.chrom.sizes) specifying a genome definition

Author(s)

Marcin Imielinski

`parse.gr` *parse.gr*

Description

quick function to parse gr from character vector IGV / UCSC style strings of format gr1;gr2;gr3 wohere each gr is of format chr:start-end[+/-]

Usage

```
parse.gr(...)
```

Arguments

`...`	arguments to parse.grl i.e. character strings in UCSC style chr:start-end[+-]

Author(s)

Marcin Imielinski

`parse.grl` *parse.grl*

Description

quick function to parse `GRangesList` from character vector IGV / UCSC style strings of format gr1;gr2;gr3 where each gr is of format chr:start-end[+/-]

Usage

```
parse.grl(x, seqlengths = hg_seqlengths())
```

Arguments

x	character vector representing a GRangesList with UCSC style coordinates (chr:start-end[+-]) representing a [signed] Granges and ";" separators within each item of x separating individaul each GRAnges
seqlengths	named integer vector representing genome (hg_seqlengths() by default)

Author(s)

Marcin Imielinski

---

ra.dedup *Deduplicates rearrangements represented by* GRangesList *objects*

Description

Determines overlaps between two or more piles of rearrangement junctions (as named or numbered arguments) +/- padding and will merge those that overlap into single junctions in the output, and then keep track for each output junction which of the input junctions it was "seen in" using logical flag meta data fields prefixed by "seen.by." and then the argument name (or "seen.by.ra" and the argument number)

Usage

```
ra.dedup(grl, pad = 500, ignore.strand = FALSE)
```

Arguments

grl	GRangesList representing rearrangements to be merged
pad	non-negative integer specifying padding
ignore.strand	whether to ignore strand (implies all strand information will be ignored, use at your own risk)

Value

GRangesList of merged junctions with meta data fields specifying which of the inputs each outputted junction was "seen.by"

Author(s)

Xiaotong Yao

ra.duplicated *Show if junctions are Deduplicated*

Description

Determines overlaps between two or more piles of rearrangement junctions (as named or numbered arguments) +/- padding and will merge those that overlap into single junctions in the output, and then keep track for each output junction which of the input junctions it was "seen in" using logical flag meta data fields prefixed by "seen.by." and then the argument name (or "seen.by.ra" and the argument number)

Usage

```
ra.duplicated(grl, pad = 500, ignore.strand = FALSE)
```

Arguments

grl	GRangesList representing rearrangements to be merged
pad	non-negative integer specifying padding
ignore.strand	whether to ignore strand (implies all strand information will be ignored, use at your own risk)

Value

GRangesList of merged junctions with meta data fields specifying which of the inputs each outputted junction was "seen.by"

Author(s)

Xiaotong Yao

ra.merge *Merges rearrangements represented by* GRangesList *objects*

Description

Determines overlaps between two or more piles of rearrangement junctions (as named or numbered arguments) +/- padding and will merge those that overlap into single junctions in the output, and then keep track for each output junction which of the input junctions it was "seen in" using logical flag meta data fields prefixed by "seen.by." and then the argument name (or "seen.by.ra" and the argument number)

Usage

```
ra.merge(..., pad = 0, ind = FALSE, ignore.strand = FALSE)
```

Arguments

...	GRangesList representing rearrangements to be merged
pad	non-negative integer specifying padding
ind	logical flag (default FALSE) specifying whether the "seen.by" fields should contain indices of inputs (rather than logical flags) and NA if the given junction is missing
ignore.strand	whether to ignore strand (implies all strand information will be ignored, use at your own risk)

Value

GRangesList of merged junctions with meta data fields specifying which of the inputs each outputted junction was "seen.by"

Examples

```
# generate some junctions
gr1 <- GRanges(1, IRanges(1:10, width = 1), strand = rep(c('+', '-'), 5))
gr2 <- GRanges(1, IRanges(4 + 1:10, width = 1), strand = rep(c('+', '-'), 5))
ra1 = split(gr1, rep(1:5, each = 2))
ra2 = split(gr2, rep(1:5, each = 2))

ram = ra.merge(ra1, ra2)
values(ram) # shows the metadata with TRUE / FALSE flags

ram2 = ra.merge(ra1, ra2, pad = 5) # more inexact matching results in more merging
values(ram2)

ram3 = ra.merge(ra1, ra2, ind = TRUE) #indices instead of flags
values(ram3)
```

---

ra.overlaps *ra.overlaps*

Description

Determines overlaps between two piles of rearrangement junctions ra1 and ra2 (each GRangesLists of signed locus pairs) against each other, returning a sparseMatrix that is T at entry ij if junction i overlaps junction j.

Usage

```
ra.overlaps(ra1, ra2, pad = 0, arr.ind = TRUE, ignore.strand = FALSE, ...)
```

Arguments

ra1	GRangesList with rearrangement set 1
ra2	GRangesList with rearrangement set 2
pad	Amount to pad the overlaps by. Larger is more permissive. Default is exact (0)

arr.ind	Default TRUE
ignore.strand	Ignore rearrangement orientation when doing overlaps. Default FALSE
...	params to be sent to gr.findoverlaps

Details

if argument pad = 0 (default) then only perfect overlap will validate, otherwise if pad>0 is given, then padded overlap is allowed o strand matters, though we test overlap of both ra1[i] vs ra2[j] and gr.flip(ra2[j])

---

rle.query *Queries an RleList representing genomic data*

Description

(ie a list whose names represent seqnames ie chromosomes, and lengths represent seqlengths) via GRanges object

Usage

```
rle.query(subject.rle, query.gr, verbose = FALSE, mc.cores = 1,
  chunksize = 1e+09)
```

Arguments

subject.rle	Rle
query.gr	TODO
verbose	Set the verbosity of the output
mc.cores	Number of cores to apply when doing chunked operation
chunksize	Number of query.gr ranges to consider in one memory chunk. 1e9

Value

Rle representing the (concatenated) vector of data (reversing order in case of negative strand input)

Note

Throws warning if seqlengths(gr) do not correspond to the lengths of the RleList components

rrbind — *Improved* rbind *for intersecting/union columns of* data.frames *or* data.tables

Description

Like rbind, but takes the intersecting columns of the data.

Usage

```
rrbind(..., union = TRUE, as.data.table = FALSE)
```

Arguments

...	Any number of data.frame or data.table objects
union	Take union of columns (and put NA's for columns of df1 not in df2 and vice versa). [TRUE]
as.data.table	Return the binded data as a data.table. [FALSE]

Value

data.frame or data.table of the rbind operation

Author(s)

Marcin Imielinski

seg2gr — *Convert GRange like data.frames into GRanges*

Description

Take data frame of ranges "segs" and converts into granges object porting over additional value columns "segs" data frame can obey any number of conventions to specify chrom, start, and end of ranges (eg $pos1, $pos2, $Start_position, $End_position) –> see "standardize_segs" for more info

Take data frame of ranges "segs" and converts into granges object porting over additional value columns "segs" data frame can obey any number of conventions to specify chrom, start, and end of ranges (eg $pos1, $pos2, $Start_position, $End_position) –> see "standardize_segs" for more info

Usage

```
seg2gr(segs, seqlengths = NULL, seqinfo = Seqinfo())

standardize_segs(seg, chr = FALSE)
```

Arguments

segs	data frame of segments with fields denoting chromosome, start, end, and other metadata (see standardized segs for seg data frame input formats)
seqlengths	seqlengths of output GRanges object
seqinfo	seqinfo of output GRanges object
segs	data frame of segments with fields denoting chromosome, start, end, and other metadata (see standardized segs for seg data frame input formats)
seqlengths	seqlengths of output GRanges object
seqinfo	seqinfo of output GRanges object

Details

standardize_segs

(data frame seg function)

Takes and returns segs data frame standardized to a single format (ie $chr, $pos1, $pos2)

if chr = TRUE will ensure "chr" prefix is added to chromossome(if does not exist)

---

si — Seqinfo *object for hg19*

Description

Seqinfo object for hg19

Format

Seqinfo

---

si2gr — *Create* GRanges *from* Seqinfo *or* BSgenome

Description

Creates a genomic ranges from seqinfo object ie a pile of ranges spanning the genome

Usage

```
si2gr(si, strip.empty = FALSE)
```

Arguments

si	Seqinfo object or a BSgenome genome
strip.empty	Don't know. [FALSE]

Value

GRanges representing the range of the input genome

Examples

```
si2gr(BSgenome.Hsapiens.UCSC.hg19::Hsapiens)
```

---

streduce *Reduce* GRanges *and* GRangesList *to miminal footprint*

Description

Shortcut for `reduce(sort(gr.stripstrand(unlist(x))))`

Usage

```
streduce(gr, pad = 0, sort = TRUE)
```

Arguments

gr	GRanges or GRangesList
pad	Expand the input data before reducing. [0]
sort	Flag to sort the output. [TRUE]

Value

GRanges object with no strand information, representing a minimal footprint

Examples

```
streduce(grl.hiC, pad=10)
streduce(example_genes, pad=1000)
```

%&% *subset x on y ranges wise ignoring strand*

Description

shortcut for x[gr.in(x,y)]

gr1

Shortcut for gr.in

gr1

Shortcut for gr.in (standard arguments)

gr1

Usage

```
x %&% ...

x %^% ...

x %^% ...
```

Arguments

x	See gr.in
...	See gr.in
x	GRanges object
...	additional arguments to gr.in
x	See gr.in
...	See gr.in

Value

subset of gr1 that overlaps gr2f

logical vector of length gr1 which is TRUE at entry i only if gr1[i] intersects at least one interval in gr2

logical vector of length gr1 which is TRUE at entry i only if gr1[i] intersects at least one interval in gr2 (strand agnostic)

Author(s)

Marcin Imielinski

Marcin Imielinski

%&&% *Subset x on y ranges wise respecting strand*

Description

shortcut for x[gr.in(x,y)]

gr1

Usage

```
x %&&% ...
```

Value

subset of gr1 that overlaps gr2

Author(s)

Marcin Imielinski

%+% *Nudge GRanges right*

Description

Operator to shift GRanges right "sh" bases

Usage

```
gr %+% ...
```

Value

shifted granges

Author(s)

Marcin Imielinski

%-% *Shift GRanges left*

Description

Operator to shift GRanges left "sh" bases

df

Usage

```
gr %-% ...
```

Value

shifted granges

Author(s)

Marcin Imielinski

%Q% *query ranges by applying an expression to ranges metadata*

Description

gr

Usage

```
## S4 method for signature 'GRanges'
x %Q% y
```

Arguments

x	GRanges to match against a query GRanges
y	GRanges with metadata to be queried

Value

subset of gr that matches query

Author(s)

Marcin Imielinski

%_% BiocGenerics::setdiff *shortcut (strand agnostic)*

Description

Shortcut for BiocGenerics::setdiff

gr1 <- GRanges(1, IRanges(10,20), strand="+") gr2 <- GRanges(1, IRanges(15,25), strand="-") gr3 <- "1:1-15" gr1 gr1

More robust and faster implementation of GenomicRangs::setdiff

Robust to common edge cases of setdiff(gr1, gr2) where gr2 ranges are contained inside gr1's (yieldings setdiffs yield two output ranges for some of the input gr1 intervals.

Usage

```
x %_% ...

gr.setdiff(query, subject, ignore.strand = TRUE, by = NULL, ...)
```

Arguments

x	GRanges object to to
...	A GRanges or a character to be parsed into a GRanges
query	GRanges object as query
subject	GRanges object as subject
max.slice	Default Inf. If query is bigger than this, chunk into smaller on different cores
verbose	Default FALSE
mc.cores	Default 1. Only works if exceeded max.slice
...	arguments to be passed to gr.findoverlaps

Value

GRanges representing setdiff of input interval

returns indices of query in subject or NA if none found

Author(s)

Marcin Imielinski

%O% *gr.val shortcut to get fractional overlap of gr1 by gr2, ignoring strand*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %O% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

fractional overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%00% *gr.val shortcut to get fractional overlap of gr1 by gr2, respecting strand*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %00% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

fractional overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%$$% *gr.val shortcut to get mean values of subject "x" meta data fields in query "y" (respects strand)*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %$$% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

gr1 with extra meta data fields populated from gr2

Author(s)

Marcin Imielinski

%NN% — *gr.val shortcut to get total numbers of intervals in gr2 overlapping with each interval in gr1, respecting strand*

Description

gr1

Usage

```
x %NN% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%N% — *gr.val shortcut to get total numbers of intervals in gr2 overlapping with each interval in gr1, ignoring strand*

Description

gr1

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %N% ...

x %$% ...
```

Arguments

x	See gr.val
...	See gr.val
x	GRanges object

Value

bases overlap of gr1 with gr2

gr1 with extra meta data fields populated from gr2

Author(s)

Marcin Imielinski

Marcin Imielinski

%oo% *gr.val shortcut to total per interval width of overlap of gr1 with gr2, respecting strand*

Description

gr1

Usage

```
x %oo% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%o% *gr.val shortcut to total per interval width of overlap of gr1 with gr2, ignoring strand*

Description

Shortcut for gr.val (using val = names(values(y)))

gr1

Usage

```
x %o% ...
```

Arguments

x	See gr.val
...	See gr.val

Value

bases overlap of gr1 with gr2

Author(s)

Marcin Imielinski

%**% *gr.findoverlaps (respects strand)*

Description

Shortcut for gr.findoverlaps

gr1

Usage

```
x %**% ...
```

Arguments

x	See gr.findoverlaps
...	See gr.findoverlaps

Value

new granges containing every pairwise intersection of ranges in gr1 and gr2 with a join of the corresponding metadata

Author(s)

Marcin Imielinski

`%*%` *Metadata join with coordinates as keys (wrapper to* `gr.findoverlaps`*)*

Description

Shortcut for gr.findoverlaps with `qcol` and `scol` filled in with all the query and subject metadata names. This function is useful for piping `GRanges` operations together. Another way to think of join of the metadata, with genomic coordinates as the keys.
Example usage:
x

Usage

```
## S4 method for signature 'GRanges,ANY'
x %*% y
```

Arguments

x	GRanges
y	GRanges

Value

`GRanges` containing every pairwise intersection of ranges in `x` and `y` with a join of the corresponding metadata

Author(s)

Marcin Imielinski

Examples

```
example_genes %*% example_dnase
```

Index