Package ‘gUtils’

March 18, 2016

Title R Package Providing Additional Capabilities and Speed for GenomicRanges Operations

Version 0.1

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)

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

License GPL-2

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

LazyData true

RoxygenNote 5.0.1

R topics documented:

`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)
```

Arguments

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

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('+', '-')))
```

`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.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`

Examples

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

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

gr.DNAase *DNAaseI hypersensitivity sites for hg19*

Description

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

Format

`GRanges`

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.

Examples

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

gr.findoverlaps — *Faster replacement for* GRanges *version of* GenomicRanges::findOverlaps

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,
  foverlaps = ifelse(is.na(as.logical(Sys.getenv("GRFO_FOVERLAPS"))), FALSE,
  as.logical(Sys.getenv("GRFO_FOVERLAPS"))) & exists("foverlaps"),
  pintersect = NA, verbose = FALSE, type = "any", by = NULL,
  return.type = "same", ...)
```

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
`foverlaps`	Use `data.table::foverlaps` instead of `IRanges::findOverlaps`. Overrules `pintersect`
`pintersect`	Use `IRanges::pintersect` function. In general this is slower, but can be much faster and with lower memory for large ranges on many different seqnames (e.g. transcriptome). Default is `FALSE` unless `length(unique(seqnames)) > 50`
`verbose`	Increase the verbosity during runtime [FALSE]
`type`	`type` argument as defined by `IRanges::findOverlaps` (`"any"`, `"start"`, `"end"`, `"within"`, `"equal"`). If value other than `"any"` is supplied, will force call to `IRanges::findOverlaps`. Otherwise, may use `IRanges::pintersect` or `data.table::foverlaps`. `["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"]`
`...`	Additional arguments sent to `IRanges::pintersect` if `pintersect = TRUE`.

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.genes` *RefSeq genes from UCSC Table Browser hg19, subsampled to 10,000 genes*

Description

RefSeq genes from UCSC Table Browser hg19, subsampled to 10,000 genes

Format

`GRanges`

`gr.in` *Faster version of* `GRanges` `over`

Description

Uses `gr.findoverlaps` for a faster `over`.

Usage

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

Arguments

`query`	Query `GRanges` pile
`subject`	Subject `GRanges` pile
`...`	Additional arguments to pass to `gr.findoverlaps`

Details

Can specify a `by` = column name in query and subject that we additionally control for a match (passed on to `gr.findoverlaps`).

Value

logical vector if query range i is found in any range in subject

gr.match — *Faster* GenomicRanges::match

Description

Faster implementation of GenomicRanges::match (uses gr.findoverlaps)

Usage

```
gr.match(query, subject, max.slice = Inf, verbose = FALSE, mc.cores = 1,
  ...)
```

Arguments

query	Query GRanges pile
subject	Subject GRanges pile
max.slice	Maximum number of ranges to consider at once [Inf]
verbose	Increase the verbosity during runtime
mc.cores	Number of cores to use, if ranges exceed max.slice
...	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)

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.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

Examples

```
## Generate a single random interval of width 10, on "chr" of length 1000
gr.rand(10, Seqinfo("1", 1000))
## Generate 5 non-overlapping regions of width 10 on hg19
library(BSgenome.Hsapiens.UCSC.hg19)
gr.rand(rep(10,5), 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, len = 100, replace = TRUE)
```

Arguments

gr	GRanges object defining the territory to sample from
k	Number of ranges to sample
len	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.

Note

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

Examples

```
## sample 5 \code{GRanges} of length 10 each from territory of RefSeq genes
gr.sample(reduce(gr.genes), k=5, len=10)
```

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 = FALSE)
```

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(gr.DNAase, width=200)
gr.start(gr.DNAase, 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())
```

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 ";")

Examples

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

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"

---

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
```

---

gr2dt *Converts* GRanges *to* data.table

Description

Converts GRanges to `data.table`

Usage

```
gr2dt(gr)
```

Arguments

gr	GRanges pile to convert to `data.table`

Value

`data.table` with seqnames, start, end, width, strand and all of the meta data. Width is end-inclusive (e.g. [6,7] width = 2)

Examples

```
gr2dt(gr.genes)
```

`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(gr.genes, gr.DNAase)

Note

Does not fill in the `Seqinfo` for the output `GRanges`

`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, ...)
```

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]
`...`	Additional parameters to be passed on to `gr.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.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

Examples

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

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)
```

---

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

Examples

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

---

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

---

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

```
## Not run: libary(BSgenome.Hsapiens.UCSC.hg19); si2gr(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(gr.genes, pad=1000)
```

Index