Package 'GenomicTuples'

April 8, 2026

Type Package

Title Representation and Manipulation of Genomic Tuples

Version 1.45.0

Date 2025-07-02

Encoding UTF-8

Description GenomicTuples defines general purpose containers for storing genomic tuples. It aims to provide functionality for tuples of genomic co-ordinates that are analogous to those available for genomic ranges in the GenomicRanges Bioconductor package.

URL www.github.com/PeteHaitch/GenomicTuples

BugReports https://github.com/PeteHaitch/GenomicTuples/issues

biocViews Infrastructure, DataRepresentation, Sequencing

VignetteBuilder knitr

Depends R (>= 4.0), GenomicRanges (>= 1.37.4), Seqinfo, S4Vectors (>= 0.17.25)

Imports methods, BiocGenerics (>= 0.21.2), Rcpp (>= 0.11.2), IRanges (>= 2.19.13), data.table, stats4, stats, utils

Suggests testthat, knitr, BiocStyle, rmarkdown, covr, GenomicAlignments, Biostrings, GenomeInfoDb

LinkingTo Rcpp

License Artistic-2.0

Collate 'AllGenerics.R' 'AllUtilities.R' 'GTuples-class.R' 'GTuples-comparison.R' 'GTuplesList-class.R' 'GenomicTuples.R' 'RcppExports.R' 'coverage-methods.R' 'findOverlaps-methods.R' 'inter-tuple-methods.R' 'intra-tuple-methods.R' 'nearest-methods.R' 'setops-methods.R' 'tile-methods.R' 'zzz.R'

RoxygenNote 7.3.2

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

git_branch devel

git_last_commit 8f1b003

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-04-07

Author Peter Hickey [aut, cre],
Marcin Cieslik [ctb],
Hervé Pagès [ctb]

Maintainer Peter Hickey <peter.hickey@gmail.com>

Contents

`GenomicTuples-package` *Representation and manipulation of genomic tuples.*

Description

GenomicTuples defines general purpose containers for storing genomic tuples. It aims to provide functionality for tuples of genomic co-ordinates that are analogous to those available for genomic ranges in the GenomicRanges Bioconductor package.

Details

Please refer to the vignettes to see how to use the **GenomicTuples** package.

Author(s)

Maintainer: Peter Hickey <peter.hickey@gmail.com>

Other contributors:

- Marcin Cieslik <marcin.cieslik@gmail.com> [contributor]
- Hervé Pagès [contributor]

References

Peter F Hickey (2016). Representation and Manipulation of Genomic Tuples in R. *JOSS*. URL http://dx.doi.org/10.21105/joss.00020

See Also

Useful links:

- www.github.com/PeteHaitch/GenomicTuples
- Report bugs at https://github.com/PeteHaitch/GenomicTuples/issues

---

`.allTuplesSortedCpp` *An internal helper function to check that each tuple is sorted in increasing order; only used for tuples of size > 2.*

Description

An internal helper function to check that each tuple is sorted in increasing order; only used for tuples of size > 2.

Usage

```
.allTuplesSortedCpp(pos1, internal_pos, posm)
```

Arguments

`pos1`	An integer vector.
`internal_pos`	An integer matrix.
`posm`	An integer vector.

Details

.allTuplesSorted is adapted from http://stackoverflow.com/a/7601857. Strict inequalities are required.

Value

TRUE if each tuple is sorted in strictly increasing order, FALSE otherwise.

.findEqual.GTuples *An internal function used by the findOverlaps,GTuples,GTuples-method when type = "equal".*

Description

An internal function used by the findOverlaps,GTuples,GTuples-method when type = "equal".

Usage

```
.findEqual.GTuples(query, subject, select, ignore.strand)
```

Arguments

query	A GTuples instance
subject	A GTuples instance
select	See findOverlaps in the IRanges package for a description of this argument.
ignore.strand	When set to TRUE, the strand information is ignored in the overlap calculations.

.GT2DT *Convert a GTuples object to a data.table.*

Description

Convert a GTuples object to a data.table.

Usage

```
.GT2DT(gt, ignore.strand = FALSE)
```

Arguments

gt	A GTuples object
ignore.strand	When set to TRUE, the strand is set to "*".

`.IPDCpp` *An internal function to compute the IPD of a GTuples; ; only used for tuples of size > 2.*

Description

An internal function to compute the IPD of a GTuples; ; only used for tuples of size > 2.

Usage

```
.IPDCpp(pos1, internal_pos, posm)
```

Arguments

`pos1`	An integer vector.
`internal_pos`	An integer matrix.
`posm`	An integer vector.

Value

An integer matrix with the same number of rows as internal_pos and number of columns equal to size - 1.

Note

This function silently coerces numeric matrices to integer matrices and does integer subtraction. *This will give unexpected results but it's not a problem for me since I only use it on integer matrices.*

`.matrixDiffWithRecycling` *Compute column-wise difference of matrices with possibly different number of rows. Do this by iterating over columns, treating them as vectors and then using R's native vector recycling.*

Description

Compute column-wise difference of matrices with possibly different number of rows. Do this by iterating over columns, treating them as vectors and then using R's native vector recycling.

Usage

```
.matrixDiffWithRecycling(x, y)
```

.pcompareGTuplesCpp *An internal function used to pcompare GTuples.*

Description

An internal function used to pcompare GTuples.

Usage

```
.pcompareGTuplesCpp(int_seqnames, int_strand, int_pos)
```

Arguments

int_seqnames	An integer vector of length n. An integer representation of the difference in seqnames of each tuple.
int_strand	An integer vector of length n. An integer representation of the difference in strand of each tuple.
int_pos	An integer matrix with n rows. Each row represents the difference in positions of each tuple.

Details

The tuples should have already been converted to integer representations, namely an integer vector for the difference in chromosome, an integer vector for the difference in strand and an integer matrix for the difference in positions.

Value

An integer vector where each element is the comparison of a pair of tuples. If the first tuple in the pair is "<" than the second tuple then the return value for that element is < 0, if the first tuple in the pair is "==" the second tuple then the return value is 0, and if the first tuple is ">" that the second tuple then the return value is > 0.

.zero_range *Check whether all elements of a numeric vector are identical (within machine precision)*

Description

Check whether all elements of a numeric vector are identical (within machine precision)

Usage

```
.zero_range(x, tol = .Machine$double.eps^0.5)
```

Arguments

x	a numeric vector.

Value

TRUE if all elements of the vector are identical (within machine precision). FALSE in all other cases, including if the vector contains any NAs.

Note

This function is based on Hadley and John's answer to http://stackoverflow.com/q/4752275. No check is made that x is a numeric vector.

---

findOverlaps-methods *Finding overlapping genomic tuples*

Description

Various methods for finding/counting overlaps between objects containing genomic tuples. This man page describes the methods that operate on GTuples and GTuplesList objects.

NOTE: The ?findOverlaps generic function is defined and documented in the **IRanges** package. The findOverlaps method for GenomicRanges and GRangesList objects are defined and documented in the **GenomicRanges** package.

GTuples and GTuplesList objects also support countOverlaps, overlapsAny, and subsetByOverlaps thanks to the default methods defined in the **IRanges** package and to the findOverlaps and countOverlaps methods defined in this package and documented below.

Usage

```
## S4 method for signature 'GTuples,GTuples'
findOverlaps(query, subject,
    maxgap = -1L, minoverlap = 0L,
    type = c("any", "start", "end", "within", "equal"),
    select = c("all", "first", "last", "arbitrary"),
    ignore.strand = FALSE)

## S4 method for signature 'GTuples,GTuples'
countOverlaps(query, subject,
    maxgap = -1L, minoverlap = 0L,
    type = c("any", "start", "end", "within", "equal"),
    ignore.strand = FALSE)
```

Arguments

query, subject	A GTuples or GTuplesList object.
type	See details below.
maxgap, minoverlap	See ?findOverlaps in the **IRanges** package for a description of these arguments.
select	When select is "all" (the default), the results are returned as a Hits object. Otherwise the returned value is an integer vector parallel to query (i.e. same length) containing the first, last, or arbitrary overlapping interval in subject, with NA indicating intervals that did not overlap any intervals in subject.
ignore.strand	When set to TRUE, the strand information is ignored in the overlap calculations.

Details

The findOverlaps-based methods involving genomic tuples, either through GTuples or GTuplesList objects, can search for *tuple-tuple*, *tuple-range* and *range-tuple* overlaps. Each of these are described below, with attention paid to the important special case of finding "equal tuple-tuple overlaps".

Equal tuple-tuple overlaps When the query and the subject are both GTuples objects and type = "equal", findOverlaps uses the seqnames (seqnames), positions (tuples,GTuples-method) and strand (strand) to determine which tuples from the query exactly match those in the subject, where a strand value of "*" is treated as occurring on both the "+" and "-" strand. An overlap is recorded when a tuple in the query and a tuple in the subject have the same sequence name, have a compatible pairing of strands (e.g. "+"/"+", "-"/"-", "*"/"+", "*"/"-", etc.), and have identical positions.

NOTE: Equal tuple-tuple overlaps can only be computed if size(query) is equal to size(subject).

Other tuple-tuple overlaps When the query and the subject are GTuples or GTuplesList objects and type = "any", "start", "end" or "within", findOverlaps treats the tuples as if they were ranges, with ranges given by $[pos_1, pos_m]$ and where m is the size,GTuples-method of the tuples. This is done via inheritance so that a GTuples (resp. GTuplesList) object is treated as a GRanges (resp. GRangesList) and the appropriate findOverlaps method is dispatched upon.

NOTE: This is the only type of overlap finding available when either the query and subject are GTuplesList objects. This is following the behaviour of findOverlaps,GRangesList,GRangesList-method that allows type = "any", "start", "end" or "within" but does not allow type = "equal".

tuple-range and range-tuple overlaps When one of the query and the subject is not a GTuples or GTuplesList objects, findOverlaps treats the tuples as if they were ranges, with ranges given by $[pos_1, pos_m]$ and where m is the size,GTuples-method of the tuples. This is done via inheritance so that a GTuples (resp. GTuplesList) object is treated as a GRanges (resp. GRangesList) and the appropriate findOverlaps method is dispatched upon.

In the context of findOverlaps, a feature is a collection of tuples/ranges that are treated as a single entity. For GTuples objects, a feature is a single tuple; while for GTuplesList objects, a feature is a list element containing a set of tuples. In the results, the features are referred to by number, which run from 1 to length(query)/length(subject).

Value

For `findOverlaps`, either a `Hits` object when `select = "all"` or an integer vector otherwise.

For `countOverlaps`, an integer vector containing the tabulated query overlap hits.

For `overlapsAny` a logical vector of length equal to the number of tuples/ranges in `query` indicating those that overlap any of the tuples/ranges in `subject`.

For `subsetByOverlaps` an object of the same class as `query` containing the subset that overlapped at least one entity in `subject`.

Author(s)

Peter Hickey for methods involving `GTuples` and `GTuplesList`. P. Aboyoun, S. Falcon, M. Lawrence, N. Gopalakrishnan, H. Pagès and H. Corrada Bravo for all the real work underlying the powerful `findOverlaps` functionality.

See Also

- Please see the package vignette for an extended discussion of overlaps involving genomic tuples, which is available by typing `vignette(topic = 'GenomicTuplesIntroduction', package = 'GenomicTuples')` at the R prompt.
- `findOverlaps`
- `findOverlaps`
- `Hits`
- `GTuples`
- `GTuplesList`
- `GRanges`
- `GRangesList`

Examples

```
## GTuples object containing 3-tuples:
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L, 30L, 30L, 35L, 30L, 30L), ncol = 3),
               strand = c('+', '-', '*', '+', '+'))

## GTuplesList object
gtl3 <- GTuplesList(A = gt3[1:3], B = gt3[4:5])

## Find equal genomic tuples:
findOverlaps(gt3, gt3, type = 'equal')
## Note that this is different to the results if the tuples are treated as
## ranges since this ignores the "internal positions" (pos2):
findOverlaps(granges(gt3), granges(gt3), type = 'equal')

## Scenarios where tuples are treated as ranges:
findOverlaps(gt3, gt3, type = 'any')
findOverlaps(gt3, gt3, type = 'start')
```

```
findOverlaps(gt3, gt3, type = 'end')
findOverlaps(gt3, gt3, type = 'within')

## Overlapping a GTuples and a GTuplesList object (tuples treated as ranges):
table(!is.na(findOverlaps(gtl3, gt3, select="first")))
countOverlaps(gtl3, gt3)
findOverlaps(gtl3, gt3)
subsetByOverlaps(gtl3, gt3)
countOverlaps(gtl3, gt3, type = "start")
findOverlaps(gtl3, gt3, type = "start")
subsetByOverlaps(gtl3, gt3, type = "start")
findOverlaps(gtl3, gt3, select = "first")
```

GTuples-class *GTuples objects*

Description

The GTuples class is a container for the genomic tuples and their associated annotations.

Details

GTuples extends GRanges as a container for genomic tuples rather than genomic ranges. GTuples is a vector of genomic locations and associated annotations. Each element in the vector is comprised of a sequence name, a tuple, a strand, and optional metadata columns (e.g. score, GC content, etc.). This information is stored in four components:

seqnames a 'factor' Rle object containing the sequence names.

tuples externally, a matrix-link object containing the tuples. Internally, an IRanges object storing the first and last position of each tuple and, if required, a matrix storing the "internal" positions of each tuple (see description of internalPos below).

strand a Rle object containing the strand information.

mcols a DataFrame object containing the metadata columns. Columns cannot be named "seqnames" "ranges", "tuples", "internalPos", "size", "strand", "seqlevels", "seqlengths", "isCircular", "start", "end", "width", or "element".

seqinfo a DataFrame object containing information about the set of genomic sequences present in the GTuples object.

Slots

Since the GTuples class extends the GRanges class it contains the seqnames, ranges, strand, elementMetadata, seqinfo and metadata. The GTuples class also contains two additional slots, size and internalPos.

size An integer. The size of the genomic tuples stored in the GTuples object.

internalPos If the size of the genomic tuples is greater than 2, internalPos is an integer matrix storing the "internal" positions of each genomic tuple. Otherwise internalPos is NULL.

Constructor

`GTuples(seqnames = Rle(), tuples = matrix(), strand = Rle("*", length(seqnames)), ..., seqlengths = NULL,`
Creates a `GTuples` object.

`seqnames` `Rle` object, character vector, or factor containing the sequence names.

`tuples` matrix object containing the positions of the tuples. The first column should refer to pos1, the second to pos2, etc.

`strand` `Rle` object, character vector, or factor containing the strand information.

`...` Optional metadata columns. These columns cannot be named "`start`", "`end`", "`width`", or "`element`". A named integer vector "`seqlength`" can be used instead of `seqinfo`.

`seqlengths` an integer vector named with the sequence names and containing the lengths (or `NA`) for each `level(seqnames)`.

`seqinfo` a `DataFrame` object containing allowed sequence names and lengths (or `NA`) for each `level(seqnames)`.

Coercion

In the code snippets below, `x` is a GTuples object.

`as.data.frame(x, row.names = NULL, optional = FALSE, ...)`**:** Creates a data.frame with columns `seqnames` (factor), `tuples` (integer), `strand` (factor), as well as the additional metadata columns stored in `mcols(x)`. Pass an explicit `stringsAsFactors=TRUE/FALSE` argument via `...` to override the default conversions for the metadata columns in `mcols(x)`.

`as.character(x, ignore.strand=FALSE)`**:** Turn GTuples object `x` into a character vector where each tuples in `x` is represented by a string in format `chr1:100,109,115:+`. If `ignore.strand` is TRUE or if *all* the ranges in `x` are unstranded (i.e. their strand is set to `*`), then all the strings in the output are in format `chr1:100,109,115`.

The names on `x` are propagated to the returned character vector. Its metadata (`metadata(x)`) and metadata columns (`mcols(x)`) are ignored.

`as.factor(x)`**:** Equivalent to

```
factor(as.character(x), levels=as.character(sort(unique(x))))
```

`as(x, "GRanges"), granges(x)`**:** Creates a `GRanges` object from a `GTuples` object. **WARNING:** This is generally a *destructive* operation because all "internal" positions will be dropped.

Accessors

In the following code snippets, `x` is a `GTuples` object.

`size(x)`**:** Get the size of the genomic tuples stored in `x`.

`length(x)`**:** Get the number of elements.

`seqnames(x)`**,** `seqnames(x) <- value`**:** Get or set the sequence names. `value` can be an `Rle` object, a character vector, or a factor.

`tuples(x)`**,** `tuples(x) <- value`**:** Get the positions of the tuples, which are returned as an integer matrix. `value` can be an integer matrix.

`ranges(x, use.mcols = FALSE)`, `ranges(x) <- value`: Get or set the ranges in the form of a CompressedIRangesList. `value` can be a IntegerRangesList object.

WARNING: The use of `ranges` with `GTuples` objects is **strongly** discouraged. It will only get/set pos_1 and pos_m of the tuples, where m is the `size` of the tuples, as these are what are stored in the "ranges" slot of a `GTuples` object.

`names(x)`, `names(x) <- value`: Get or set the names of the elements.

`strand(x)`, `strand(x) <- value`: Get or set the strand. `value` can be an `Rle` object, character vector, or factor.

`mcols(x, use.names=FALSE)`, `mcols(x) <- value`: Get or set the metadata columns. If `use.names=TRUE` and the metadata columns are not `NULL`, then the names of `x` are propagated as the row names of the returned `DataFrame` object. When setting the metadata columns, the supplied value must be `NULL` or a data.frame-like object (i.e. `DataFrame` or `data.frame`) object holding element-wise metadata.

`elementMetadata(x)`, `elementMetadata(x) <- value`, `values(x)`, `values(x) <- value`: Alternatives to `mcols` functions. Their use is discouraged.

`seqinfo(x)`, `seqinfo(x) <- value`: Get or set the information about the underlying sequences. `value` must be a `DataFrame` object.

`seqlevels(x)`, `seqlevels(x, force=FALSE) <- value`: Get or set the sequence levels. `seqlevels(x)` is equivalent to `seqlevels(seqinfo(x))` or to `levels(seqnames(x))`, those 2 expressions being guaranteed to return identical character vectors on a `GTuples` object. `value` must be a character vector with no `NA`s. See `?seqlevels` for more information.

`seqlengths(x)`, `seqlengths(x) <- value`: Get or set the sequence lengths. `seqlengths(x)` is equivalent to `seqlengths(seqinfo(x))`. `value` can be a named non-negative integer or numeric vector eventually with `NA`s.

`isCircular(x)`, `isCircular(x) <- value`: Get or set the circularity flags. `isCircular(x)` is equivalent to `isCircular(seqinfo(x))`. `value` must be a named logical vector eventually with `NA`s.

`genome(x)`, `genome(x) <- value`: Get or set the genome identifier or assembly name for each sequence. `genome(x)` is equivalent to `genome(seqinfo(x))`. `value` must be a named character vector eventually with `NA`s.

`seqlevelsStyle(x)`, `seqlevelsStyle(x) <- value`: Get or set the seqname style for `x`. See the `seqlevelsStyle` generic getter and setter in the **GenomeInfoDb** package for more information.

`score(x)`, `score(x) <- value`: Get or set the "score" column from the element metadata.

Tuples methods

In the following code snippets, x is a `GTuples` object. **WARNING**: The preferred setter is `tuples(x) <- value` and the use of `start(x) <- value`, `end(x) <- value` and `width(x) <- value` is discouraged.

`start(x)`, `start(x) <- value`: Get or set pos_1 of the tuples. **WARNING**: The use of `width(x) <- value` is discouraged; instead, construct the tuples as the appropriate integer matrix, `mvalue`, and use `tuples(x) <- mvalue`.

`end(x)`, `end(x) <- value`: Get or set pos_m of the tuples, where m is the `size` of the tuples. **WARNING**: The use of `end(x) <- value` is discouraged; instead, construct the tuples as the appropriate integer matrix, `mvalue`, and use `tuples(x) <- mvalue`.

`IPD(x)`: Get the intra-pair distances (`IPD`). `IPD` is only defined for tuples with `size` > 1. The `IPD` of a tuple with `size`= m is the vector of intra-pair distances, $(pos_2 - pos_1, \ldots, pos_m - pos_{m-1})$.

`width(x)`, `width(x) <- value`: Get or set $pos_m - pos_1$ of the tuples, where m is the `size` of the tuples. If using `width(x) <- value`, pos_1 is held fixed and pos_m is altered. **WARNING**: The use of `width(x) <- value` is discouraged; instead, construct the tuples as the appropriate integer matrix, `mvalue`, and use `tuples(x) <- mvalue`.

Splitting and Combining

In the following code snippets, x is a `GTuples` object.

`append(x, values, after = length(x))`: Inserts the values into `x` at the position given by after, where `x` and `values` are of the same class.

`c(x, ...)`: Combines `x` and the `GTuples` objects in `...` together. Any object in `...` must belong to the same class as `x`, or to one of its subclasses, or must be `NULL`. The result is an object of the same class as `x`.

`c(x, ..., ignore.mcols=FALSE)`: If the `GTuples` objects have metadata columns (represented as one `DataFrame` per object), each such `DataFrame` must have the same columns in order to combine successfully. In order to circumvent this restraint, you can pass in an `ignore.mcols=TRUE` argument which will combine all the objects into one and drop all of their metadata columns.

`split(x, f, drop=FALSE)`: Splits `x` according to `f` to create a `GTuplesList` object. If `f` is a list-like object then `drop` is ignored and `f` is treated as if it was `rep(seq_len(length(f)), sapply(f, length))`, so the returned object has the same shape as `f` (it also receives the names of `f`). Otherwise, if `f` is not a list-like object, empty list elements are removed from the returned object if `drop` is `TRUE`.

Subsetting

In the following code snippets, x is a `GTuples` object.

`x[i, j]`, `x[i, j] <- value`: Get or set elements `i` with optional metadata columns `mcols(x)[,j]`, where `i` can be missing; an `NA`-free logical, numeric, or character vector; or a 'logical' `Rle` object.

`x[i,j] <- value`: Replaces elements `i` and optional metadata columns `j` with `value`.

`head(x, n = 6L)`: If `n` is non-negative, returns the first `n` elements of the `GTuples` object. If `n` is negative, returns all but the last `abs(n)` elements of the `GTuples` object.

`rep(x, times, length.out, each)`: Repeats the values in `x` through one of the following conventions:

`times` Vector giving the number of times to repeat each element if of length `length(x)`, or to repeat the whole vector if of length 1.

`length.out` Non-negative integer. The desired length of the output vector.

`each` Non-negative integer. Each element of `x` is repeated each times.

`subset(x, subset)`: Returns a new object of the same class as `x` made of the subset using logical vector subset, where missing values are taken as `FALSE`.

`tail(x, n = 6L)`: If `n` is non-negative, returns the last `n` elements of the `GTuples` object. If `n` is negative, returns all but the first `abs(n)` elements of the `GTuples` object.

`window(x, start = NA, end = NA, width = NA, frequency = NULL, delta = NULL, ...)`**:** Extracts the subsequence window from the `GTuples` object using:

`start`**,** `end`**,** `width` The start, end, or width of the window. Two of the three are required.

`frequency`**,** `delta` Optional arguments that specify the sampling frequency and increment within the window.

In general, this is more efficient than using `"["` operator.

`window(x, start = NA, end = NA, width = NA, keepLength = TRUE) <- value`**:** Replaces the subsequence window specified on the left (i.e. the subsequence in `x` specified by `start`, `end` and `width`) by `value`. `value` must either be of class `class(x)`, belong to a subclass of `class(x)`, be coercible to `class(x)`, or be `NULL`. If `keepLength` is `TRUE`, the elements of `value` are repeated to create a `GTuples` object with the same number of elements as the width of the subsequence window it is replacing. If `keepLength` is `FALSE`, this replacement method can modify the length of `x`, depending on how the length of the left subsequence window compares to the length of `value`.

`x$name`**,** `x$name <- value`**:** Shortcuts for `mcols(x)$name` and `mcols(x)$name <- value`, respectively. Provided as a convenience, from `GRanges` as the result of strong popular demand. Note that those methods are not consistent with the other `$` and `$<-` methods in the `IRanges`/`GenomicRanges` infrastructure, and might confuse some users by making them believe that a `GRanges` object can be manipulated as a data.frame-like object. Therefore we recommend using them only interactively, and we discourage their use in scripts or packages. For the latter, use `mcols(x)$name` and `mcols(x)$name <- value`, instead of `x$name` and `x$name <- value`, respectively.

Other methods

`show(x)`**:** By default the show method displays 5 head and 5 tail elements. This can be changed by setting the global options `showHeadLines` and `showTailLines`. If the object length is less than (or equal to) the sum of these 2 options plus 1, then the full object is displayed. Note that these options also affect the display of `GRanges` objects (defined in the **GenomicRanges** package), `GAlignments` and `GAlignmentPairs` objects (defined in the **GenomicAlignments** package), as well as other objects defined in the **IRanges** and **Biostrings** packages (e.g. `IRanges` and `DNAStringSet` objects).

Author(s)

Peter Hickey

See Also

`GTuplesList-class`, `seqinfo`, `Vector`, `Rle`, `DataFrame`, `GRanges`, `intra-tuple-methods`, `findOverlaps-methods`, `nearest-methods`,

Examples

```
## Create example 4-tuples
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
gt4 <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                              c(1, 3, 2, 4)),
```

```
                tuples = matrix(c(1:10, 2:11, 3:12, 4:13), ncol = 4),
                strand = Rle(strand(c("-", "+", "*", "+", "-")),
                             c(1, 2, 2, 3, 2)),
                score = 1:10, GC = seq(1, 0, length = 10), seqinfo = seqinfo)
gt4

## Summarizing elements
table(seqnames(gt4))
sum(width(gt4))
summary(mcols(gt4)[,"score"])

## Renaming the underlying sequences
seqlevels(gt4)
seqlevels(gt4) <- sub("chr", "Chrom", seqlevels(gt4))
gt4
seqlevels(gt4) <- sub("Chrom", "chr", seqlevels(gt4)) # revert

## Combining objects
gt4_a <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                                c(1, 3, 2, 4)),
                tuples = matrix(c(1:10, 21:30, 31:40, 41:50), ncol = 4),
                strand = Rle(strand(c("-", "+", "*", "+", "-")),
                             c(1, 2, 2, 3, 2)),
                score = 1:10, seqinfo = seqinfo)

gt4_b <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                                c(1, 3, 2, 4)),
                tuples = matrix(c(101:110, 121:130, 131:140, 141:150),
                                ncol = 4),
                strand = Rle(strand(c("-", "+", "*", "+", "-")),
                             c(1, 2, 2, 3, 2)),
                score = 1:10, seqinfo = seqinfo)

some_gt4 <- c(gt4_a, gt4_b)

## all_gt4 <- c(gt4, gt4_a, gt4_b) ## (This would fail)
all_gt4 <- c(gt4, gt4_a, gt4_b, ignore.mcols=TRUE)

## The number of lines displayed in the 'show' method
## are controlled with two global options.
options("showHeadLines" = 7)
options("showTailLines" = 2)
all_gt4

## Revert to default values
options("showHeadLines"=NULL)
options("showTailLines"=NULL)

## Get the size of the tuples stored in the GTuples object
size(gt4)

## Get the tuples
tuples(gt4)
```

```
  ## Get the matrix of intra-pair distances (IPD)
  IPD(all_gt4)

  ## Can't combine genomic tuples of different sizes
  gt1 <- GTuples('chr1', matrix(30:40))
  gt1
  ## Not run:
  ## Returns error
  c(gt4, gt1)

## End(Not run)
```

GTuples-comparison *Comparing and ordering genomic tuples*

Description

Methods for comparing and ordering the elements in one or more GTuples objects.

Usage

```
## duplicated()
## ------------

## S4 method for signature 'GTuples'
duplicated(x, incomparables = FALSE, fromLast = FALSE)

## match() and selfmatch()
## -------

## S4 method for signature 'GTuples,GTuples'
match(x, table, nomatch = NA_integer_,
      incomparables = NULL, ignore.strand = FALSE)
## S4 method for signature 'GTuples'
selfmatch(x, ignore.strand = FALSE, ...)

## order() and related methods
## ---------------------------------------------------------------------------

## S4 method for signature 'GTuples'
order(..., na.last = TRUE, decreasing = FALSE, method = c("auto", "shell", "radix"))

## S4 method for signature 'GTuples'
sort(x, decreasing = FALSE, ignore.strand = FALSE, by)

## S4 method for signature 'GTuples'
rank(x, na.last = TRUE,
```

```
    ties.method = c("average", "first", "random", "max", "min"))

## S4 method for signature 'GTuples'
is.unsorted(x, na.rm=FALSE, strictly=FALSE, ignore.strand = FALSE)

## Generalized element-wise (aka "parallel") comparison of 2 GTuples
## objects
## ------------------------------------------------------------------------

## S4 method for signature 'GTuples,GTuples'
pcompare(x, y)
```

Arguments

x, table, y	GTuples objects.
incomparables	Not supported.
fromLast, method, nomatch	See ?`GenomicRanges-comparison` in the **GenomicRanges** package for a description of these arguments.
ignore.strand	Whether or not the strand should be ignored when comparing 2 genomic tuples.
...	One or more GTuples objects. The GTuples objects after the first one are used to break ties
na.last	Ignored.
decreasing	TRUE or FALSE.
ties.method	A character string specifying how ties are treated. Only "first" is supported for now.
by	An optional formula that is resolved against as.env(x); the resulting variables are passed to order to generate the ordering permutation.
na.rm	logical. Should missing values be removed before checking? WARNING: This currently has no effect and is ignored.
strictly	logical indicating if the check should be for *strictly* increasing values.

Details

Two elements of a GTuples object (i.e. two genomic tuples) are considered equal if and only if they are on the same underlying sequence and strand, and have the same positions (tuples). duplicated() and unique() on a GTuples object are conforming to this.

The "natural order" for the elements of a GTuples object is to order them (a) first by sequence level, (b) then by strand, (c) then by $pos_1, \ldots, pos_m$. This way, the space of genomic tuples is totally ordered.

order(), sort(), is.unsorted(), and rank() on a GTuples object are using this "natural order".

Also ==, !=, <=, >=, < and > on GTuples objects are using this "natural order".

pcompare(x, y): Performs "generalized range-wise comparison" of x and y, that is, returns an integer vector where the i-th element is a code describing how the i-th element in x is qualitatively positioned relatively to the i-th element in y.

A code that is < 0, = 0, or > 0, corresponds to x[i] < y[i], x[i] == y[i], or x[i] > y[i], respectively.

WARNING: These predefined codes are not as detailed as those for IPosRanges-comparison. Specifically, only the sign of the code matters, not the actual value.

`match(x, table, nomatch = NA_integer_)`**:** Returns an integer vector of the length of x, containing the index of the first matching range in table (or nomatch if there is no matching range) for each tuple in x.

`duplicated(x, fromLast = FALSE, method = c("hash", "base"))`**:** Determines which elements of x are equal to elements with smaller subscripts, and returns a logical vector indicating which elements are duplicates. See duplicated in the **base** package for more details.

`unique(x, fromLast = FALSE, method = c("hash", "base"))`**:** Removes duplicate tuples from x. See unique in the **base** package for more details.

`x %in% table`**:** A shortcut for finding the ranges in x that match any of the tuples in table. Returns a logical vector of length equal to the number of tuples in x.

`findMatches(x, table)`**:** An enhanced version of match that returns all the matches in a Hits object.

`countMatches(x, table)`**:** Returns an integer vector of the length of x containing the number of matches in table for each element in x.

`order(...)`**:** Returns a permutation which rearranges its first argument (a GTuples object) into ascending order, breaking ties by further arguments. See order in the **BiocGenerics** package for more information.

`sort(x)`**:** Sorts x. See sort in the **base** package for more details.

`rank(x, na.last = TRUE, ties.method = c("average", "first", "random", "max", "min"))`**:** Returns the sample ranks of the tuples in x. See rank in the **base** package for more details.

Value

For pcompare: see Details section above.

For selfmatch: an integer vector of the same length as x.

For duplicated, unique, and %in%: see ?BiocGenerics::duplicated, ?BiocGenerics::unique, and ?`%in%`.

For findMatches: a Hits object by default (i.e. if select="all").

For countMatches: an integer vector of the length of x containing the number of matches in table for each element in x.

For sort: see ?BiocGenerics::sort.

Author(s)

Peter Hickey

See Also

- The GTuples class.
- GenomicRanges-comparison in the **GRanges** package for comparing and ordering genomic ranges.

- intra-tuple-methods for intra-tuple transformations.
- findOverlaps-methods for finding overlapping genomic ranges.

Examples

```
## GTuples object containing 3-tuples:
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L, 30L, 30L, 35L, 30L, 30L), ncol = 3),
               strand = c('+', '-', '*', '+', '+'))
gt3 <- c(gt3, rev(gt3[3:5]))

## ---------------------------------------------------------------------
## A. ELEMENT-WISE (AKA "PARALLEL") COMPARISON OF 2 GTuples OBJECTS
## ---------------------------------------------------------------------
gt3[2] == gt3[2]  # TRUE
gt3[2] == gt3[5]  # FALSE
gt3 == gt3[4]
gt3 >= gt3[3]

## ---------------------------------------------------------------------
## B. duplicated(), unique()
## ---------------------------------------------------------------------
duplicated(gt3)
unique(gt3)

## ---------------------------------------------------------------------
## C. match(), %in%
## ---------------------------------------------------------------------
table <- gt3[2:5]
match(gt3, table)
match(gt3, table, ignore.strand = TRUE)

## ---------------------------------------------------------------------
## D. findMatches(), countMatches()
## ---------------------------------------------------------------------
findMatches(gt3, table)
countMatches(gt3, table)

findMatches(gt3, table, ignore.strand = TRUE)
countMatches(gt3, table, ignore.strand = TRUE)

gt3_levels <- unique(gt3)
countMatches(gt3_levels, gt3)

## ---------------------------------------------------------------------
## E. order() AND RELATED METHODS
## ---------------------------------------------------------------------
is.unsorted(gt3)
order(gt3)
sort(gt3)
is.unsorted(sort(gt3))
```

```
is.unsorted(gt3, ignore.strand=TRUE)
gt3_2 <- sort(gt3, ignore.strand=TRUE)
is.unsorted(gt3_2)  # TRUE
is.unsorted(gt3_2, ignore.strand=TRUE)  # FALSE

## TODO (TODO copied from GenomicRanges): Broken. Please fix!
#sort(gt3, by = ~ seqnames + start + end) # equivalent to (but slower than) above

score(gt3) <- rev(seq_len(length(gt3)))

## TODO (TODO copied from GenomicRanges): Broken. Please fix!
#sort(gt3, by = ~ score)

rank(gt3)

## ---------------------------------------------------------------------
## F. GENERALIZED ELEMENT-WISE COMPARISON OF 2 GTuples OBJECTS
## ---------------------------------------------------------------------
pcompare(gt3[3], gt3)
```

GTuplesList-class *GTuplesList objects*

Description

The GTuplesList class is a container for storing a collection of GTuples objects. It is derived from GRangesList.

Constructor

GTuplesList(...): Creates a GTuplesList object using GTuples objects supplied in ....

Accessors

In the following code snippets, x is a GTuplesList object.

length(x): Get the number of list elements.

names(x), names(x) <- value: Get or set the names on x.

elementNROWS(x): Get the length of each of the list elements.

isEmpty(x): Returns a logical indicating either if the GTuplesList has no elements or if all its elements are empty.

seqnames(x), seqnames(x) <- value: Get or set the sequence names in the form of an RleList. value can be an RleList or CharacterList object.

tuples(x), tuples(x) <- value: Get or set the tuples in the form of a SimpleList of integer matrices. value can be a a single integer matrix.

`ranges(x, use.mcols = FALSE)`, `ranges(x) <- value`**:** Get or set the ranges in the form of a `CompressedIRangesList`. `value` can be a `IntegerRangesList` object.

WARNING: The use of `ranges` with `GTuplesList` objects is **strongly** discouraged. It will only get/set pos_1 and pos_m of the tuples, where m is the `size` of the tuples, as these are what are stored in the "ranges" slot of the `GTuples` objects.

`strand(x)`, `strand(x) <- value`**:** Get or set the strand in the form of an `RleList`. `value` can be an `RleList`, `CharacterList` or single character. `value` as a single character converts all ranges in `x` to the same `value`; for selective strand conversion (i.e., mixed "+" and "-") use `RleList` or `CharacterList`.

`mcols(x, use.names=FALSE)`, `mcols(x) <- value`**:** Get or set the metadata columns. `value` can be `NULL`, or a data.frame-like object (i.e. `DataFrame` or data.frame) holding element-wise metadata.

`elementMetadata(x)`, `elementMetadata(x) <- value`, `values(x)`, `values(x) <- value`**:** Alternatives to `mcols` functions. Their use is discouraged.

`seqinfo(x)`, `seqinfo(x) <- value`**:** Get or set the information about the underlying sequences. `value` must be a Seqinfo object.

`seqlevels(x)`, `seqlevels(x, force=FALSE) <- value`**:** Get or set the sequence levels. `seqlevels(x)` is equivalent to `seqlevels(seqinfo(x))` or to `levels(seqnames(x))`, those 2 expressions being guaranteed to return identical character vectors on a `GTuplesList` object. `value` must be a character vector with no NAs. See ?`seqlevels` for more information.

`seqlengths(x)`, `seqlengths(x) <- value`**:** Get or set the sequence lengths. `seqlengths(x)` is equivalent to `seqlengths(seqinfo(x))`. `value` can be a named non-negative integer or numeric vector eventually with NAs.

`isCircular(x)`, `isCircular(x) <- value`**:** Get or set the circularity flags. `isCircular(x)` is equivalent to `isCircular(seqinfo(x))`. `value` must be a named logical vector eventually with NAs.

`genome(x)`, `genome(x) <- value`**:** Get or set the genome identifier or assembly name for each sequence. `genome(x)` is equivalent to `genome(seqinfo(x))`. `value` must be a named character vector eventually with NAs.

`seqlevelsStyle(x)`, `seqlevelsStyle(x) <- value`**:** Get or set the seqname style for `x`. See the seqlevelsStyle generic getter and setter in the **GenomeInfoDb** package for more information.

`score(x)`, `score(x) <- value`**: Get or set the "score" metadata column.**

Tuples methods

In the following code snippets, x is a `GTuplesList` object.

WARNING: The preferred setter is `tuples(x) <- value` and the use of `start(x) <- value`, `end(x) <- value` and `width(x) <- value` is discouraged.

`start(x)`, `start(x) <- value`**:** Get or set pos_1 of the `tuples`. **WARNING**: The use of `start(x) <- value` is discouraged; instead, construct the tuples as the appropriate `List` of integer matrices, `mvalue`, and use `tuples(x) <- mvalue`.

`end(x)`, `end(x) <- value`**:** Get or set pos_m of the tuples, where m is the `size` of the tuples. **WARNING**: The use of `end(x) <- value` is discouraged; instead, construct the tuples as the appropriate `List` of integer matrices, `mvalue`, and use `tuples(x) <- mvalue`.

`IPD(x)`: Get the intra-pair distances (`IPD`) in the form of a `SimpleList` of integer matrices. `IPD` is only defined for tuples with `size` > 1. The `IPD` of a tuple with `size`= m is the vector of intra-pair distances, $(pos_2 - pos_1, \ldots, pos_m - pos_{m-1})$.

`width(x)`, `width(x) <- value`: Get or set $pos_m - pos_1$ of the tuples, where m is the `size` of the tuples. If using `width(x) <- value`, pos_1 is held fixed and pos_m is altered. **WARNING**: The use of `width(x) <- value` is discouraged; instead, instead, construct the tuples as the appropriate `List` of integer matrices, `mvalue`, and use `tuples(x) <- mvalue`.

Coercion

In the code snippets below, `x` is a GTuplesList object.

`as.data.frame(x, row.names = NULL, optional = FALSE, ..., value.name = "value", use.outer.mcols = FALSE, g`
Coerces `x` to a `data.frame`. See as.data.frame on the `List` man page for details (?`List`).

`as.list(x, use.names = TRUE)`: Creates a list containing the elements of `x`.

`as(x, "GRangesList")`: Creates a `GRangesList` object from a `GTuplesList` object. **WARNING:** This is generally a *destructive* operation, as the original `GTuplesList` may not be re-creatable.

Subsetting

In the following code snippets, `x` is a `GTuplesList` object.

`x[i, j]`, `x[i, j] <- value`: Get or set elements `i` with optional metadata columns `mcols(x)[,j]`, where `i` can be missing; an NA-free logical, numeric, or character vector; a 'logical' Rle object, or an AtomicList object.

`x[[i]]`, `x[[i]] <- value`: Get or set element `i`, where `i` is a numeric or character vector of length 1.

`x$name`, `x$name <- value`: Get or set element `name`, where `name` is a name or character vector of length 1.

`head(x, n = 6L)`: If `n` is non-negative, returns the first `n` elements of the GTuplesList object. If `n` is negative, returns all but the last `abs(n)` elements of the GTuplesList object.

`rep(x, times, length.out, each)`: Repeats the values in `x` through one of the following conventions:

`times` Vector giving the number of times to repeat each element if of length `length(x)`, or to repeat the whole vector if of length 1.

`length.out` Non-negative integer. The desired length of the output vector.

`each` Non-negative integer. Each element of `x` is repeated `each` times.

`subset(x, subset)`: Returns a new object of the same class as `x` made of the subset using logical vector `subset`, where missing values are taken as `FALSE`.

`tail(x, n = 6L)`: If `n` is non-negative, returns the last `n` elements of the GTuples object. If `n` is negative, returns all but the first `abs(n)` elements of the GTuples object.

Combining

In the code snippets below, x is a GTuplesList object.

`c(x, ...)`: Combines x and the GTuplesList objects in ... together. Any object in ... must belong to the same class as x, or to one of its subclasses, or must be NULL. The result is an object of the same class as x.

`append(x, values, after = length(x))`: Inserts the values into x at the position given by after, where x and values are of the same class.

`unlist(x, recursive = TRUE, use.names = TRUE)`: Concatenates the elements of x into a single GTuples object.

Looping

In the code snippets below, x is a GTuplesList object.

`endoapply(X, FUN, ...)`: Similar to lapply, but performs an endomorphism, i.e. returns an object of `class(X)`.

`lapply(X, FUN, ...)`: Like the standard lapply function defined in the base package, the lapply method for GTuplesList objects returns a list of the same length as X, with each element being the result of applying FUN to the corresponding element of X.

`Map(f, ...)`: Applies a function to the corresponding elements of given GTuplesList objects.

`mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)`: Like the standard mapply function defined in the base package, the mapply method for GTuplesList objects is a multivariate version of sapply.

`mendoapply(FUN, ..., MoreArgs = NULL)`: Similar to mapply, but performs an endomorphism across multiple objects, i.e. returns an object of `class(list(...)[[1]])`.

`Reduce(f, x, init, right = FALSE, accumulate = FALSE)`: Uses a binary function to successively combine the elements of x and a possibly given initial value.

f A binary argument function.

init An R object of the same kind as the elements of x.

right A logical indicating whether to proceed from left to right (default) or from right to left.

nomatch The value to be returned in the case when "no match" (no element satisfying the predicate) is found.

`sapply(X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)`: Like the standard sapply function defined in the base package, the sapply method for GTuplesList objects is a user-friendly version of lapply by default returning a vector or matrix if appropriate.

Author(s)

Peter Hickey for GTuplesList definition and methods. P. Aboyoun & H. Pagès for all the real work underlying the powerful GRangesList class and methods.

See Also

GTuples-class seqinfo, GRangesList, Vector, IntegerRangesList, RleList, DataFrameList, findOverlaps-methods

Examples

```
## Construction of GTuplesList of 4-tuples with GTuplesList():
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
gt4 <- GTuples(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"),
                              c(1, 3, 2, 4)),
               tuples = matrix(c(1:10, 2:11, 3:12, 4:13), ncol = 4),
               strand = Rle(strand(c("-", "+", "*", "+", "-")),
                            c(1, 2, 2, 3, 2)),
               score = 1:10, GC = seq(1, 0, length = 10), seqinfo = seqinfo)
gtl4 <- GTuplesList(A = gt4[1:4], B = gt4[5:10])
gtl4

## Summarizing elements:
elementNROWS(gtl4)
table(seqnames(gtl4))

## Extracting subsets:
gtl4[seqnames(gtl4) == "chr1", ]
gtl4[seqnames(gtl4) == "chr1" & strand(gtl4) == "+", ]

## Renaming the underlying sequences:
seqlevels(gtl4)
seqlevels(gtl4) <- sub("chr", "Chrom", seqlevels(gtl4))
gtl4

## Coerce to GRangesList ("internal positions" information is lost):
as(gtl4, "GRangesList")

## Get the size of the tuples stored in the GTuplesList object
size(gtl4)

## Get the tuples
tuples(gtl4)

## Get the matrix of intra-pair distances (IPD)
IPD(gtl4)

## Can't combine genomic tuples of different sizes
gt1 <- GTuples('chr1', matrix(30:40))
gt1
## Not run:
## Returns error
GTuplesList(A = gt4, gt1)

## End(Not run)
```

`intra-tuple-methods` *Intra-tuple transformations of a GTuples or GTuplesLists object*

Description

This man page documents intra-tuple transformations of a GTuples or a GTuplesList object.

WARNING: These are not exactly the same as the intra-range methods defined in the **GenomicRanges** package (?GenomicRanges::intra-range-methods) or in the **IRanges** package (?IRanges::intra-range-meth

Usage

```
## S4 method for signature 'GTuples'
shift(x, shift = 0L, use.names = TRUE)
## S4 method for signature 'GTuplesList'
shift(x, shift = 0L, use.names = TRUE)

## S4 method for signature 'GTuples'
trim(x, use.names = TRUE)
```

Arguments

x	A GTuples or GTuplesList object.
shift, use.names	See ?`intra-range-methods`.

Details

shift: behaves like the shift method for GRanges objects, except that any internalPos are also shifted. See ?`intra-range-methods` for further details of the shift method.

trim: trims out-of-bound tuples located on non-circular sequences whose length is not NA.

Value

See Details section above.

Author(s)

Peter Hickey for methods involving GTuples and GTuplesList. P. Aboyoun and V. Obenchain <vobencha@fhcrc.org> for all the real work underlying the powerful intra-range methods.

See Also

- GTuples and GTuplesList objects.
- The intra-range-methods man page in the **GenomicRanges** package.

Examples

```
## ---------------------------------------------------------------------
## A. ON A GTuples OBJECT
## ---------------------------------------------------------------------
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L, 30L, 30L, 35L, 30L, 30L), ncol = 3),
```

```
                strand = c('+', '-', '*', '+', '+'))
gt3

shift(gt3, 10)

## ---------------------------------------------------------------------
## B. ON A GTuplesList OBJECT
## ---------------------------------------------------------------------
gtl3 <- GRangesList(A = gt3, B = rev(gt3))
gtl3

shift(gtl3, IntegerList(10, 100))
```

---

nearest-methods *Finding the nearest genomic tuple/range neighbour*

Description

The nearest, precede, follow, distance and distanceToNearest methods for GTuples objects and subclasses.

NOTE: These methods treat the tuples as if they were ranges, with ranges given by $[pos_1, pos_m]$ and where m is the size,GTuples-method of the tuples. This is done via inheritance so that a GTuples object is treated as a GRanges and the appropriate method is dispatched upon.

Usage

```
## S4 method for signature 'GTuples,GTuples'
precede(x, subject, select = c("arbitrary", "all"),
        ignore.strand = FALSE, ...)
## S4 method for signature 'GTuples,missing'
precede(x, subject, select = c("arbitrary", "all"),
        ignore.strand = FALSE, ...)

## S4 method for signature 'GTuples,GTuples'
follow(x, subject, select = c("arbitrary", "all"),
       ignore.strand=FALSE, ...)
## S4 method for signature 'GTuples,missing'
follow(x, subject, select = c("arbitrary", "all"),
       ignore.strand = FALSE, ...)

## S4 method for signature 'GTuples,GTuples'
nearest(x, subject, select = c("arbitrary", "all"),
        ignore.strand = FALSE, ...)
## S4 method for signature 'GTuples,missing'
nearest(x, subject, select = c("arbitrary", "all"),
        ignore.strand = FALSE, ...)
```

```
## S4 method for signature 'GTuples,GTuples'
distanceToNearest(x, subject, ignore.strand = FALSE,
                  ...)
## S4 method for signature 'GTuples,missing'
distanceToNearest(x, subject, ignore.strand = FALSE,
                  ...)

## S4 method for signature 'GTuples,GTuples'
distance(x, y, ignore.strand = FALSE, ...)
```

Arguments

x	The query GTuples instance.
subject	The subject GTuples instance within which the nearest neighbours are found. Can be missing, in which case x is also the subject.
y	For the distance method, a GTuples or GRanges instance. Cannot be missing. If x and y are not the same length, the shortest will be recycled to match the length of the longest.
select	Logic for handling ties. By default, all methods select a single tuple/range (arbitrary for nearest, the first by order in subject for precede, and the last for follow). When select = "all" a Hits object is returned with all matches for x. If x does not have a match in subject the x is not included in the Hits object.
ignore.strand	A logical indicating if the strand of the input tuples/ranges should be ignored. When TRUE, strand is set to '+'.
...	Additional arguments for methods.

Details

nearest: Performs conventional nearest neighbour finding. Returns an integer vector containing the index of the nearest neighbour tuple/range in subject for each range in x. If there is no nearest neighbour NA is returned. For details of the algorithm see the man page in IRanges, ?nearest.

precede: For each range in x, precede returns the index of the tuple/range in subject that is directly preceded by the tuple/range in x. Overlapping tuples/ranges are excluded. NA is returned when there are no qualifying tuples/ranges in subject.

follow: The opposite of precede, follow returns the index of the tuple/range in subject that is directly followed by the tuple/range in x. Overlapping tuples/ranges are excluded. NA is returned when there are no qualifying tuples/ranges in subject.

Orientation and strand for precede and follow: Orientation is 5' to 3', consistent with the direction of translation. Because positional numbering along a chromosome is from left to right and transcription takes place from 5' to 3', precede and follow can appear to have 'opposite' behavior on the + and - strand. Using positions 5 and 6 as an example, 5 precedes 6 on the + strand but follows 6 on the - strand.

The table below outlines the orientation when tuples/ranges on different strands are compared. In general, a feature on * is considered to belong to both strands. The single exception is when both x and subject are * in which case both are treated as +.

```
        x  |  subject  |  orientation
      -----+-----------+----------------
a)     +  |  +        |  --->
b)     +  |  -        |  NA
c)     +  |  *        |  --->
d)     -  |  +        |  NA
e)     -  |  -        |  <---
f)     -  |  *        |  <---
g)     *  |  +        |  --->
h)     *  |  -        |  <---
i)     *  |  *        |  --->  (the only situation where * arbitrarily means +)
```

`distanceToNearest`**:** Returns the distance for each tuple/range in `x` to its nearest neighbour in the `subject`.

`distance`**:** Returns the distance for each tuple/range in `x` to the range in `y`. The behaviour of `distance` has changed in Bioconductor 2.12. See the man page `?distance` in IRanges for details.

Value

For `nearest`, `precede` and `follow`, an integer vector of indices in `subject`, or a`Hits` if `select = "all"`.

For `distanceToNearest`, a `Hits` object with a column for the `query` index (from), `subject` index (to) and the `distance` between the pair.

For `distance`, an integer vector of distances between the tuples/ranges in `x` and `y`.

Author(s)

Peter Hickey for methods involving `GTuples`. P. Aboyoun and V. Obenchain <vobencha@fhcrc.org> for all the real work underlying the powerful nearest methods.

See Also

- The GTuples and GRanges classes.
- `GenomicRanges` and `GRanges` classes in the **GenomicRanges** package.
- The`IPosRanges` class in the **IRanges** package.
- The `Hits` class in the **S4Vectors** package.
- The nearest-methods man page in the **GenomicRanges** package.
- findOverlaps-methods for finding just the overlapping ranges.

Examples

```
## ------------------------------------------------------------
## precede() and follow()
## ------------------------------------------------------------
query <- GTuples("A", matrix(c(5L, 20L, 6L, 21L), ncol = 2), strand = "+")
subject <- GTuples("A", matrix(c(rep(c(10L, 15L), 2), rep(c(11L, 16L), 2)),
                               ncol = 2),
```

```
                        strand = c("+", "+", "-", "-"))
precede(query, subject)
follow(query, subject)

strand(query) <- "-"
precede(query, subject)
follow(query, subject)

## ties choose first in order
query <- GTuples("A", matrix(c(10L, 11L), ncol = 2), c("+", "-", "*"))
subject <- GTuples("A", matrix(c(rep(c(5L, 15L), each = 3),
                                 rep(c(6L, 16L), each = 3)), ncol = 2),
                        rep(c("+", "-", "*"), 2))
precede(query, subject)
precede(query, rev(subject))

## ignore.strand = TRUE treats all ranges as '+'
precede(query[1], subject[4:6], select="all", ignore.strand = FALSE)
precede(query[1], subject[4:6], select="all", ignore.strand = TRUE)

## -------------------------------------------------------------
## nearest()
## -------------------------------------------------------------
## When multiple tuples overlap an "arbitrary" tuple is chosen
query <- GTuples("A", matrix(c(5L, 15L), ncol = 2))
subject <- GTuples("A", matrix(c(1L, 15L, 5L, 19L), ncol = 2))
nearest(query, subject)

## select = "all" returns all hits
nearest(query, subject, select = "all")

## Tuples in 'x' will self-select when 'subject' is present
query <- GTuples("A", matrix(c(1L, 10L, 6L, 15L), ncol = 2))
nearest(query, query)

## Tuples in 'x' will not self-select when 'subject' is missing
nearest(query)

## -------------------------------------------------------------
## distance(), distanceToNearest()
## -------------------------------------------------------------
## Adjacent, overlap, separated by 1
query <- GTuples("A", matrix(c(1L, 2L, 10L, 5L, 8L, 11L), ncol = 2))
subject <- GTuples("A", matrix(c(6L, 5L, 13L, 10L, 10L, 15L), ncol = 2))
distance(query, subject)

## recycling
distance(query[1], subject)

query <- GTuples(c("A", "B"), matrix(c(1L, 5L, 2L, 6L), ncol = 2))
distanceToNearest(query, subject)
```

`tuples-squeezers` *Squeeze the tuples out of a tuples-based object*

Description

S4 generic functions for squeezing the tuples out of a tuples-based object. Similar to the S4 generic functions for squeezing the ranges out of a ranged-based object, see granges and grglist.

`gtuples` returns them as a GTuples object, and `gtlist` as a GTuplesList object.

Usage

```
gtuples(x, use.mcols=FALSE, ...)
gtlist(x, use.mcols=FALSE, ...)
```

Arguments

`x`	A tuples-based object.
`use.mcols`	`TRUE` or `FALSE` (the default). Whether the metadata columns on `x` (accessible with `mcols(x)`) should be propagated to the returned object or not.
`...`	Additional arguments, for use in specific methods.

Details

The **MethylationTuples** (`https://github.com/PeteHaitch/MethylationTuples`) package defines and document methods for various types of tuples-based objects. Other Bioconductor packages might as well.

Note that these functions can be seen as a specific kind of *object getters* as well as functions performing coercion.

Value

A GTuples object for `gtuples`.

A GTuplesList object for `gtlist`.

If `x` is a vector-like object, the returned object is expected to be *parallel* to `x`, that is, the i-th element in the output corresponds to the i-th element in the input. If `x` has names on it, they're propagated to the returned object. If `use.mcols` is TRUE and `x` has metadata columns on it (accessible with `mcols(x)`), they're propagated to the returned object.

Author(s)

Peter Hickey

See Also

- GTuples and GTuplesList objects.

Examples

```
## See ?MethPat in the MethylationTuples package (GitHub-only package) for some
## examples.
```

`Undefined methods` *Undefined methods*

Description

These are methods defined for `GRanges` and `GRangesList` objects that have no well-defined equivalent for `GTuples` or `GTuplesList`. Therefore, I have explicitly written methods for these that return errors when called.

Examples

```
gt3 <- GTuples(seqnames = c('chr1', 'chr1', 'chr1', 'chr1', 'chr2'),
               tuples = matrix(c(10L, 10L, 10L, 10L, 10L, 20L, 20L, 20L, 25L,
                                 20L, 30L, 30L, 35L, 30L, 30L), ncol = 3),
               strand = c('+', '-', '*', '+', '+'))
## Not run:
# Will return errors
narrow(gt3)
reduce(gt3)

## End(Not run)
```

Index