Package 'InteractionSet'

April 23, 2025

Version 1.36.0

Date 2023-05-13

Title Base Classes for Storing Genomic Interaction Data

Depends GenomicRanges, SummarizedExperiment

Imports methods, Matrix, Rcpp, BiocGenerics, S4Vectors (>= 0.27.12), IRanges, GenomeInfoDb

Suggests testthat, knitr, rmarkdown, BiocStyle

LinkingTo Rcpp

biocViews Infrastructure, DataRepresentation, Software, HiC

Description Provides the GInteractions, InteractionSet and ContactMatrix objects and associated methods for storing and manipulating genomic interaction data from Hi-C and ChIA-PET experiments.

License GPL-3

NeedsCompilation yes

SystemRequirements C++11

VignetteBuilder knitr

Collate AllGenerics.R AllClasses.R ContactMatrix-methods.R GInteractions-methods.R InteractionSet-methods.R GRanges-methods.R getset.R swapAnchors.R pairs.R conversion.R linearize.R boundingBox.R distances.R overlaps.R linkOverlaps.R matching.R updateObject.R

RoxygenNote 7.1.1

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

git_branch RELEASE_3_21

git_last_commit 9bcce6d

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-04-23

Author Aaron Lun [aut, cre],
Malcolm Perry [aut],
Elizabeth Ing-Simmons [aut]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

`boundingBox` *Get the bounding box*

Description

Computing a minimum bounding box for groups of pairwise interactions.

Usage

```
## S4 method for signature 'GInteractions'
boundingBox(x, f)

# Note, the same call is used for the InteractionSet method.
```

Arguments

x	A GInteractions or InteractionSet object.
f	A factor or vector of length equal to that of x, indicating the group to which each pairwise interaction belongs.

Details

For any group of pairwise interactions, the minimum bounding box is the smallest rectangle in the interaction space that contains all interactions in the group. Each side of the box has coordinates spanning the most extreme anchor regions on the corresponding chromosome. This is often useful for summarizing clusters of interactions.

Grouping of interactions is specified using f, where interactions in x with the same level of f are considered to be in the same group. If f is not specified, all interactions in x are assumed to be in a single group (named as "1"). An error will be raised if a group spans multiple chromosomes for either the first or second anchor regions.

The function returns a GInteractions object containing the coordinates of the bounding boxes for all groups. Each interaction represents a bounding box for a group, where the anchor regions represent the sides of the box. Entries are named according to the levels of f, in order to specify which bounding box corresponds to which group.

It is recommended to run swapAnchors prior to computing the bounding box for intra-chromosomal groups. If all anchor1 >= anchor2 or all anchor1 <= anchor2, all interactions will lie on one side of the diagonal of the intra-chromosomal interaction space. This results in the smallest possible minimum bounding box, which will only increase in size if interactions are placed on the other side of the diagonal. Alternatively, users can specify a StrictGInteractions object as an input into x, in which anchor1 <= anchor2 is enforced automatically.

Value

A GInteractions object containing the coordinates of each bounding box.

Author(s)

Aaron Lun

See Also

GInteractions-class, InteractionSet-class

Examples

```
example(GInteractions, echo=FALSE)

# Making up a sensible grouping.
gi <- sort(gi)
all.chrs <- as.character(seqnames(regions(gi)))
f <- paste0(all.chrs[anchors(gi, type="first", id=TRUE)], ".",
            all.chrs[anchors(gi, type="second", id=TRUE)])
```

```
boundingBox(gi, f)
boundingBox(swapAnchors(gi), f)

# Fails for multiple chromosomes
try(out <- boundingBox(gi))
in.A <- f=="chrA.chrA"
out <- boundingBox(gi[in.A])
```

ContactMatrix accessors

ContactMatrix accessors

Description

Methods to get and set fields in an ContactMatrix object.

Usage

```
## S4 method for signature 'ContactMatrix'
anchors(x, type="both", id=FALSE)
## S4 method for signature 'ContactMatrix'
anchorIds(x, type="both")
## S4 replacement method for signature 'ContactMatrix'
anchorIds(x, type="both") <- value

## S4 method for signature 'ContactMatrix'
regions(x)
## S4 replacement method for signature 'ContactMatrix'
regions(x) <- value
## S4 replacement method for signature 'ContactMatrix'
replaceRegions(x) <- value
## S4 replacement method for signature 'ContactMatrix'
appendRegions(x) <- value
## S4 method for signature 'ContactMatrix'
reduceRegions(x)

## S4 method for signature 'ContactMatrix'
show(x)
## S4 method for signature 'ContactMatrix'
as.matrix(x)
## S4 replacement method for signature 'ContactMatrix'
as.matrix(x) <- value
## S4 method for signature 'ContactMatrix'
t(x)

## S4 method for signature 'ContactMatrix'
dim(x)
## S4 method for signature 'ContactMatrix'
```

```
dimnames(x)
## S4 replacement method for signature 'ContactMatrix'
dimnames(x) <- value
## S4 method for signature 'ContactMatrix'
length(x)
```

Arguments

x	A ContactMatrix object.
type	a string specifying which anchors are to be extracted or replaced.
id	a scalar logical indicating whether indices or GRanges should be returned.
value	For anchorIds<-, a list of two integer vectors when type="both". The first and second vectors must have length equal to the number of rows and columns of x, respectively. For type="row" or "column", only one vector needs to be supplied corresponding to either the rows or columns. For regions<-, a GRanges object of length equal to that of regions(x). For newRegions<-, a GRanges object that is a superset of all entries in regions(x) involved in interactions. For appendRegions<-, a GRanges of any length containing additional regions. For as.matrix<-, a matrix-like object of the same dimensions as that in the matrix slot. For dimnames<-, a list of two character vectors corresponding to the row and column names, respectively. These can also be passed separately via rownames<- and colnames<-.

Details

The return value of anchors varies depending on type and id:

- If id=FALSE, a GRanges object is returned containing the regions specified by the anchor1 or anchor2 slots in x, for type=="row" or "column", respectively.
- If id=FALSE and type="both", a list is returned with two entries row and column, containing regions specified by anchor1 and anchor2 respectively.
- If id=TRUE, the integer vectors in the anchor1 or anchor2 slots of x are returned directly, depending on type. A list of length two is returned for type="both", containing both of these vectors.

Note that anchorIds is equivalent to calling anchors with id=TRUE.

Replacement in anchorIds<- can only be performed using anchor indices. If type="both", a list of two integer vectors is required in value, specifying the indices of the row- and column-wise interacting regions in x. If type="row" or "column", an integer vector is required to replace the existing row- or column-wise indices in the anchor1 or anchor2 slot, respectively.

For regions, a GRanges is returned equal to the regions slot in x. For regions<-, the GRanges in value is used to replace the regions slot. Resorting of the replacement GRanges is performed automatically, along with re-indexing of the anchors. In addition, the input GRanges must be of the same length as the existing object in the regions slot. The newRegions replacement method can take variable length GRanges, but requires that the replacement contain (at least) all ranges

contained in anchors(x). The appendRegions replacement method appends extra intervals to the existing regions slot of x. The reduceRegions method removes unused entries in the regions slot, to save memory – see reduceRegions,GInteractions-method for more details.

The show method will print out various details of the object, such as the dimensions of the matrix slot and the length of the regions slot. The as.matrix method will return the value of the matrix slot, containing a matrix-like object of interaction data. Replacement with a matrix-like object of the same dimensions can be performed using the as.matrix<- function. The t method will transpose the matrix, i.e., switch the rows and columns (and switch the vectors in the anchor1 and anchor2 slots).

The dim method will return a vector of length 2, containing the dimensions of the matrix slot. The dimnames method will return a list of length 2, containing the row and column names of matrix (these can be modified with the dimnames<- method). The length method will return an integer scalar corresponding to the total number of entries in the matrix slot.

As the ContactMatrix class inherits from the Annotated class, additional metadata can be stored in the metadata slot. This can be accessed or modified with metadata,Annotated-method.

Value

For the getters, values in various slots of x are returned, while for the setters, the slots of x are modified accordingly – see Details.

Author(s)

Aaron Lun

See Also

ContactMatrix-class

Examples

```
example(ContactMatrix, echo=FALSE) # Generate a nice object.
show(x)

# Various matrix methods:
as.matrix(x)
t(x)

nrow(x)
ncol(x)
length(x)

# Accessing anchor ranges or indices:
anchors(x)
anchors(x, type="row")
anchors(x, id=TRUE)

anchors(x, id=TRUE, type="row")
anchors(x, id=TRUE, type="column")
```

```
# Modifying anchor indices:
nregs <- length(regions(x))
anchorIds(x) <- list(sample(nregs, nrow(x), replace=TRUE),
                     sample(nregs, ncol(x), replace=TRUE))
anchors(x, id=TRUE, type="row")
anchors(x, id=TRUE, type="column")

# Accessing or modifying regions:
regions(x)
regions(x)$score <- runif(length(regions(x)))

new.ranges <- c(regions(x), resize(regions(x), fix="center", width=50))
try(regions(x) <- new.ranges) # Fails
replaceRegions(x) <- new.ranges # Succeeds

length(regions(x))
appendRegions(x) <- GRanges("chrA", IRanges(5:10+1000, 1100+5:10), score=runif(6))
length(regions(x))

reduceRegions(x)

# Setting metadata
metadata(x)$name <- "I am a contact matrix"
metadata(x)
```

`ContactMatrix class` *ContactMatrix class and constructors*

Description

The ContactMatrix class contains a matrix where rows and columns represent genomic loci. Each entry of the matrix contains information about the interaction between the loci represented by the corresponding row/column, e.g., contact frequencies. Coordinates of the loci are also contained within this class.

Usage

```
## S4 method for signature 'ANY,numeric,numeric,GRanges'
ContactMatrix(matrix, anchor1, anchor2, regions, metadata=list())

## S4 method for signature 'ANY,GRanges,GRanges,GenomicRanges_OR_missing'
ContactMatrix(matrix, anchor1, anchor2, regions, metadata=list())

## S4 method for signature 'missing,missing,missing,GenomicRanges_OR_missing'
ContactMatrix(matrix, anchor1, anchor2, regions, metadata=list())
```

Arguments

`matrix`	Any matrix-like object containing interaction data.

anchor1, anchor2
: Either a pair of numeric vectors containing indices to regions or a pair of GRanges objects. In both cases, anchor1 and anchor2 should have lengths equal to the number of rows and columns in matrix, respectively.

regions
: A GRanges object containing the coordinates of the interacting regions. This argument is optional for InteractionSet,ANY,GRanges,GRanges-method.

metadata
: A list containing experiment-wide metadata - see ?Annotated for more details.

Details

The ContactMatrix class inherits from the Annotated class, with several additional slots:

matrix: A matrix-like object.

anchor1: An integer vector specifying the index of the first interacting region.

anchor2: An integer vector specifying the index of the second interacting region.

regions: A sorted GRanges object containing the coordinates of all interacting regions.

Each entry of anchor1 corresponds to a row in matrix, while each entry of anchor2 corresponds to a column. Each entry of matrix represents an interaction between the corresponding entries in anchor1 and anchor2, i which point to the relevant coordinates in regions for each locus.

ContactMatrix objects can be constructed by specifying numeric vectors as anchor1 and anchor2 in the ContactMatrix function. These vectors will define the regions corresponding to the rows and columns of the matrix. Specifically, each value of the vector acts as an index to specify the relevant coordinates from regions. This means that the range of entries must lie within [1, length(regions)].

Alternatively, ContactMatrix objects can be constructed by directly supplying the GRanges of the interacting loci in ContactMatrix. If regions is not specified, this will be constructed automatically from the two sets of supplied GRanges. If regions is supplied, exact matching will be performed to identify the indices in regions corresponding to the regions in the supplied GRanges. Missing values are not tolerated and will cause an error to be raised.

Both methods will return an ContactMatrix object containing all of the specified information. Sorting of regions is also performed automatically, with re-indexing of all anchor indices to preserve the correct pairings between regions.

Value

For the constructors, a ContactMatrix object is returned.

Choosing between matrix classes

The ContactMatrix class provides support for any matrix-like object that implements dim, rbind, cbind, [and t methods. The choice of class depends on the type of data and the intended application. Some of the common choices are described in more detail here:

- Base matrices are simple to generate and are most efficient for dense data. This is often sufficient for most use cases where small regions of the interaction space are being examined.

- Sparse matrices from the **Matrix** package are useful for large areas of the interaction space where most entries are zero. This reduces memory usage compared to a dense representation (though conversely, is less efficient for dense data). Note that all numeric values are coerced to double-precision types, which may take up more memory than a direct integer representation. Another issue is how missing values should be interpreted in the sparseMatrix – see ?inflate for more details.
- Packed symmetric matrices from the **Matrix** package provide some memory savings for symmetric regions of the interaction space.
- Delayed or HDF5-backed matrices from the **DelayedArray** and **HDF5Array** packages allow very large matrices to be represented without loading into memory.

Author(s)

Aaron Lun

See Also

ContactMatrix-access, ContactMatrix-subset, ContactMatrix-sort, Annotated-class, Matrix-class

Examples

```
set.seed(1000)
N <- 30
all.starts <- sample(1000, N)
all.ends <- all.starts + round(runif(N, 5, 20))
all.regions <- GRanges(rep(c("chrA", "chrB"), c(N-10, 10)),
    IRanges(all.starts, all.ends))

Nr <- 10
Nc <- 20
all.anchor1 <- sample(N, Nr)
all.anchor2 <- sample(N, Nc)
counts <- matrix(rpois(Nr*Nc, lambda=10), Nr, Nc)
x <- ContactMatrix(counts, all.anchor1, all.anchor2, all.regions)

# Equivalent construction:
ContactMatrix(counts, all.regions[all.anchor1],
    all.regions[all.anchor2])
ContactMatrix(counts, all.regions[all.anchor1],
    all.regions[all.anchor2], all.regions)

# Also works directly with Matrix objects.
counts2 <- Matrix::Matrix(counts)
x2 <- ContactMatrix(counts2, all.anchor1, all.anchor2, all.regions)
counts2 <- as(counts2, "dgCMatrix")
x2 <- ContactMatrix(counts2, all.anchor1, all.anchor2, all.regions)
```

ContactMatrix distances

Compute linear distances from ContactMatrix objects

Description

Methods to compute linear distances between pairs of interacting regions in a ContactMatrix object.

Usage

```
## S4 method for signature 'ContactMatrix'
pairdist(x, type="mid")

## S4 method for signature 'ContactMatrix'
intrachr(x)
```

Arguments

x	A ContactMatrix object.
type	A character string specifying the type of distance to compute. See ?pairdist,InteractionSet-method for possible values.

Details

pairdist,ContactMatrix-method will return a matrix of integer (or, if type="intra", logical) values. Each entry of this matrix specifies the distance between the interacting loci that are represented by the corresponding row and column. If type="intra", each entry indicates whether the corresponding interaction is intra-chromosomal. Running intrachr(x) is equivalent to pairdist(x, type="intra") for any ContactMatrix object x. See pairdist,InteractionSet-method for more details on the type of distances that can be computed.

Value

An integer or logical matrix of the same dimensions as x, containing the specified distances.

Author(s)

Aaron Lun

See Also

ContactMatrix-class, pairdist,InteractionSet-method

Examples

```
example(ContactMatrix, echo=FALSE)

pairdist(x)
pairdist(x, type="gap")
pairdist(x, type="span")
pairdist(x, type="diag")

intrachr(x)
```

`ContactMatrix overlaps` *Find overlaps between GRanges and a ContactMatrix*

Description

Find overlaps between a set of linear intervals in a GRanges object, and the set of regions representing the rows or columns of a ContactMatrix.

Usage

```
## S4 method for signature 'ContactMatrix,GRanges'
overlapsAny(query, subject, maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    ignore.strand=TRUE)

## S4 method for signature 'ContactMatrix,GRangesList'
overlapsAny(query, subject, maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    ignore.strand=TRUE, use.region="both")

# The same call is used for the methods taking GRangesList, GInteractions and
# InteractionSet objects as 'subject'. For brevity, these will not be listed.
```

Arguments

`query`	A ContactMatrix object.
`subject`	A GRanges, GRangesList, GInteractions or InteractionSet object.
`maxgap, minoverlap, type`	See ?`findOverlaps` in the **GenomicRanges** package.
`ignore.strand`	See ?`findOverlaps` in the **GenomicRanges** package.
`use.region`	A string specifying how the interacting regions are to be matched to row/column regions.

Details

When subject is a GRanges, overlaps are identified between the row regions of query and the regions in subject. This is repeated for the column regions of query. A list of two logical vectors is returned, specifying the rows and columns in query that are overlapped by any region in subject. These vectors can be directly used to subset query. Alternatively, they can be used in outer to generate a logical matrix for masking – see Examples.

For all other classes of subject, two-dimensional overlaps are identified. A logical matrix is returned indicating which entries in the ContactMatrix have overlaps with the specified interactions. For any given entry, an overlap is only considered if the regions for the corresponding row and column both overlap anchor regions in the subject. See ?"findOverlaps,GInteractions,GInteractions-method" for more details.

If use.region="both", overlaps between any row/column region and the first/second interacting region of subject are considered. If use.region="same", only overlaps between row regions and the first interacting regions, or between column regions and the second interacting regions are considered. If use.region="reverse", only overlaps between row regions and the second interacting regions, or between row regions and the first interacting regions are considered.

Value

For overlapsAny,ContactMatrix,GRanges-method, a named list of two logical vectors is returned specifying the rows of columns of query overlapped by subject.

For the other overlapsAny methods, a logical matrix is returned indicating which entries in query are overlapped by subject.

Author(s)

Aaron Lun

See Also

ContactMatrix-class, findOverlaps

Examples

```
example(ContactMatrix, echo=FALSE)

of.interest <- resize(sample(regions(x), 2), width=1, fix="center")
olap <- overlapsAny(x, of.interest)
olap
x[olap$row,]
x[,olap$column]
x[olap$row,olap$column]

keep <- outer(olap$row, olap$column, "|") # OR mask
temp <- as.matrix(x)
temp[!keep] <- NA

keep <- outer(olap$row, olap$column, "&") # AND mask
temp <- as.matrix(x)
```

```
temp[!keep] <- NA

# Two dimensional overlaps.
pairing <- GRangesList(first=regions(x), second=regions(x))
olap <- overlapsAny(x, pairing)
olap
olap <- overlapsAny(sort(x), pairing) # A bit prettier
olap
```

ContactMatrix sorting *ContactMatrix sorting and ordering*

Description

Methods to sort and order ContactMatrix objects, based on the anchor indices. Also, methods to remove duplicate rows or columns in each ContactMatrix.

Usage

```
## S4 method for signature 'ContactMatrix'
order(..., na.last=TRUE, decreasing=FALSE)
## S4 method for signature 'ContactMatrix'
sort(x, decreasing=FALSE, ...)
## S4 method for signature 'ContactMatrix'
duplicated(x, incomparables=FALSE, fromLast=FALSE, ...)
## S4 method for signature 'ContactMatrix'
unique(x, incomparables=FALSE, fromLast=FALSE, ...)
```

Arguments

...	For sort,ContactMatrix-method, one or more ContactMatrix objects with the same dimensions. Otherwise, ignored in all other methods.
x	A ContactMatrix object.
na.last	A logical scalar indicating whether NA values should be ordered last. This should not be relevant as anchor indices should be finite.
decreasing	A logical scalar indicating whether sorting should be performed in decreasing order.
incomparables	A logical scalar, ignored.
fromLast	A logical scalar indicating whether the last entry of a repeated set in x should be considered as a non-duplicate.

Details

Sorting is performed based on the anchor indices of the ContactMatrix object. Rows are ordered for increasing values of the `anchor1` slot, while columns are ordered for increasing values of the `anchor2` slot. This equates to ordering by the coordinates directly, as the GRanges in the `regions` slot is always sorted. Based on this, `sort,ContactMatrix-method` will return a sorted copy of `x` with permuted rows/columns in increasing order. This can be set to decreasing order by specifying `decreasing=TRUE`.

`order,ContactMatrix-method` returns a list of 2 integer vectors. The first vector contains the permutation to rearrange the rows of `x` in increasing order, while the second vector does the same for the columns of `x`. If multiple objects are supplied to `order`, ordering will be computed using anchor indices from successive objects. In other words, ordering will be performed using anchor indices from the first object; any rows with the same `anchor1` or columns with the same `anchor2` will be split using the corresponding indices in the second object; and so on.

`duplicated,ContactMatrix-method` returns a list of two logical vectors. The first vector indicates whether rows are duplicated, based on identical values in the `anchor1` slot. The second does the same for columns based on the `anchor2` slot. For a set of duplicated rows or columns, the first occurrence of that row/column is marked as the non-duplicate if `fromLast=FALSE`, and the last entry otherwise.

`unique,ContactMatrix-method` returns an ContactMatrix object where all duplicate rows and columns have been removed from `x`. This is equivalent to subsetting based on the non-duplicate rows and columns defined in `duplicated,ContactMatrix-method`.

Note that sorting and duplicate identification only use the anchor indices. The values of the `matrix` slot are *not* used in distinguishing between rows or columns with the same index.

Value

For `sort` and `unique`, a ContactMatrix object is returned with sorted or unique rows/columns, respectively.

For `order`, a list of two integer vectors is returned containing row/column permutations.

For `duplicated`, a list of logical vectors is returned specifying which rows/columns are duplicated.

Author(s)

Aaron Lun

See Also

`ContactMatrix-class`

Examples

```
example(ContactMatrix, echo=FALSE)

anchors(x)
x2 <- sort(x)
x2
anchors(x2)
```

```
# Can also order them.
o <- order(x)
o
stopifnot(all.equal(x[o$row,o$column], x2))

# Checking duplication.
duplicated(x)
duplicated(rbind(x, x))
stopifnot(all.equal(unique(x), unique(rbind(x, x))))
```

ContactMatrix subsetting

ContactMatrix subsetting and combining

Description

Methods to subset or combine ContactMatrix objects.

Usage

```
### Subsetting

## S4 method for signature 'ContactMatrix,ANY,ANY'
x[i, j, ..., drop=TRUE]
## S4 replacement method for signature 'ContactMatrix,ANY,ANY,ContactMatrix'
x[i, j] <- value
## S4 method for signature 'ContactMatrix'
subset(x, i, j)

### Combining

## S4 method for signature 'ContactMatrix'
cbind(..., deparse.level=1)
## S4 method for signature 'ContactMatrix'
rbind(..., deparse.level=1)
```

Arguments

x	A ContactMatrix object.
i, j	A vector of subscripts, indicating the rows and columns to be subsetted for i and j, respectively.
...	For cbind, rbind and c, ... contains ContactMatrix objects to be combined. Otherwise, this argument is ignored during subsetting.
drop	A logical scalar, ignored by [,ContactMatrix,ANY,ANY-method.
value	A ContactMatrix object with dimensions equal to the length of the two subscripts (or the corresponding dimensions of x, if either subscript is missing.
deparse.level	An integer scalar; see ?base::cbind for a description of this argument.

Details

Subsetting of ContactMatrix objects behaves like that for standard matrices. Rows and columns can be selected and rearranged, with concomitant changes to the `anchor1` and `anchor2` slots. All subsetting operations will return an ContactMatrix with the specified rows and columns. However, note that the value of `regions` will not be modified by subsetting.

`cbind` will combines objects with the same rows but different columns. Errors will occur if the row regions are not identical between objects (i.e., must have same values in the slots for `regions` and `anchor1`). Conversely, `rbind` will combines objects with the same columns but different rows. Again, errors will occur if the columns are not identical (i.e., must have same values in the slots for `regions` and `anchor2`).

If subsetted anchors are required, see `?"interaction-subset"` for why subsetting should be done before calling the `anchors` method.

Value

A subsetted or combined ContactMatrix object.

Author(s)

Aaron Lun

See Also

`ContactMatrix-class`

Examples

```
example(ContactMatrix, echo=FALSE)

# Subsetting:
x[1:5,]
x[,10:15]
x[1:5,10:15]

x2 <- x
x2[1:5,] <- x[6:10,]
as.matrix(x2[,1]) <- 20

# Combining
cbind(x, x)
rbind(x, x)
```

`Convert classes` *Convert between classes*

Description

Inflate a GInteractions or InteractionSet into a ContactMatrix, or deflate a ContactMatrix to an InteractionSet.

Usage

```
## S4 method for signature 'GInteractions'
inflate(x, rows, columns, fill=TRUE, swap=TRUE, sparse=FALSE, ...)

## S4 method for signature 'InteractionSet'
inflate(x, rows, columns, assay=1L, sample=1L, fill, swap=TRUE, sparse=FALSE, ...)

## S4 method for signature 'ContactMatrix'
deflate(x, collapse=TRUE, extract, use.zero, use.na, ...)
```

Arguments

`x`	A GInteractions or InteractionSet object for `inflate`, or a ContactMatrix object for `deflate`.
`rows, columns`	An integer, logical or character vector, a GRanges object or `NULL`, indicating the regions of interest to be used as the rows or columns of the ContactMatrix.
`assay`	A numeric scalar indicating the assay of the InteractionSet object, from which values are extracted to fill the ContactMatrix.
`sample`	A numeric scalar indicating the sample (i.e., column) of the assay to extract values to fill the ContactMatrix.
`fill`	A vector (usually logical or numeric) of length equal to `nrow(x)`, containing values with which to fill the ContactMatrix. If specified, this overrides extraction of assay values for `inflate,InteractionSet-method`.
`swap`	A logical scalar indicating whether filling should also be performed after swapping anchor indices.
`sparse`	A logical scalar indicating whether the inflated matrix should use a sparseMatrix representation.
`collapse`	A logical scalar indicating whether duplicated interactions should be removed from `x` prior to deflation.
`extract`	A logical vector or matrix indicating which entries of `x` to convert into pairwise interactions.
`use.zero, use.na`	A logical scalar indicating whether to convert zero or `NA` entries into pairwise interactions.
`...`	For `inflate`, additional arguments to pass to `overlapsAny` when `rows` or `columns` is a GRanges. For `deflate`, additional arguments to pass to the InteractionSet constructor.

Value

For `inflate`, a ContactMatrix is returned.

For `deflate`, an InteractionSet object is returned.

Inflating to a ContactMatrix

The `inflate` method will return a ContactMatrix where the rows and columns correspond to specified regions of interest in `rows` and `columns`. Regions can be specified by supplying an object of various types:

- If it is an integer vector, it is assumed to refer to intervals in the `regions` slot of the input object `x`. Values of the vector need not be sorted or unique, but must lie within `[1, regions(x)]`.
- If it is a logical vector, it will subset to retain intervals in `regions(x)` that are `TRUE`.
- If it is a character vector, it is assumed to contain the names of the reference sequences of interest (i.e., chromosome names).
- If it is a GRanges object, `overlapsAny` will be called to identify the overlapping intervals of `regions(x)`.
- If it is `NULL`, all regions in `regions(x)` will be used to construct that dimension of the ContactMatrix.

For the GInteractions method, values in the matrix are filled based on user-supplied values in `fill`. Each element of `fill` corresponds to an interaction in `x` and is used to set the matrix entry at the matching row/column. Some entries of the matrix will correspond to pairwise interactions that are not present in `x` - these are filled with `NA` values.

By default, filling is repeated after swapping the anchor indices. This means that the value of the matrix at (1, 2) will be the same as that at (2, 1), i.e., the matrix is symmetric around the diagonal of the interaction space. However, if `swap=FALSE`, filling is performed so that the first and second anchor indices correspond strictly to rows and columns, respectively. This may be preferable if the order of the anchors contains some relevant information. In all cases, if duplicated interactions are present in `x` (and redundant permutations, when `swap=TRUE`), one will be arbitrarily chosen to fill the matrix.

For the InteractionSet `inflate` method, entries in the matrix are filled in based on the values in the first sample of the first assay when `fill` is missing. For more complex `x`, values from different assays and samples can be extracted using the `assay` and `sample` arguments. Note that if `fill` is specified, it will override any extraction of values from the assays.

If `sparse=TRUE`, `inflate` will return a ContactMatrix containing a sparseMatrix in the `matrix` slot. Here, entries without a corresponding interaction in `x` are set to zero, not `NA`. See below for some considerations when interpreting zeroes and `NA`s in contact matrices.

The default `fill=TRUE` has the effect of producing a logical sparse matrix in the output ContactMatrix, indicating which pairs of regions were present in `x`.

Deflating from a ContactMatrix

The `deflate` method will return an InteractionSet where each relevant entry in the ContactMatrix is converted into a pairwise interaction. Relevant entries are defined as those that are non-zero, if `use.zero` is `FALSE`; and non-`NA`, if `use.na` is `FALSE`. If `x` contains a sparseMatrix representation,

the former is set to FALSE while the latter is set to TRUE, if either are not specified. For all other matrices, use.zero=TRUE and use.na=FALSE by default.

If extract is specified, this overrides all values of use.zero and use.na. A typical application would be to deflate a number of ContactMatrix objects with the same extract matrix. This ensures that the resulting InteractionSet objects can be easily combined with cbind, as the interactions are guaranteed to be the same. Otherwise, different interactions may be extracted depending on the presence of zero or NA values.

The values of all matrix entries are stored as a one-sample assay, with each value corresponding to its pairwise interaction after conversion. Duplicate interactions are removed by default, along with redundant permutations of the anchor indices. These can be included in the returned object by setting collapse=FALSE. This setting will also store the pairs as a GInteractions object, rather than using the default StrictGInteractions object where duplicates are not stored.

Additional arguments can be used to specify the colData and metadata, which are stored in the ContactMatrix itself.

Interpreting zeroes in a sparse matrix

Storing data as a sparseMatrix may be helpful as it is more memory-efficient for sparse areas of the interaction space. However, users should keep in mind that the zero values in the sparseMatrix may not represent zeroes in fill. The majority of these values are likely to be zero just because there was no corresponding interaction in x to set it to a non-zero value.

Whether or not this is a problem depends on the application. For example, if fill represents count data and only interactions with non-zero counts are stored in x, then setting all other entries to zero is sensible. However, in other cases, it is not appropriate to fill entries corresponding to missing interactions with zero. If fill represents, e.g., log-fold changes, then setting missing entries to a value of zero will be misleading. One could simply ignore zeroes altogether, though this will also discard entries that are genuinely zero.

These problems are largely avoided with the default dense matrices, where missing entries are simply set to NA.

Author(s)

Aaron Lun

See Also

InteractionSet-class, GInteractions-class, ContactMatrix-class, sparseMatrix-class

Examples

```
example(InteractionSet, echo=FALSE)

inflate(iset, 1:10, 1:10)
inflate(iset, 1:10, 1:10, sparse=TRUE)
inflate(iset, 1:10, 1:5+10)
inflate(iset, "chrA", 1:5+10)
inflate(iset, "chrA", "chrB")
inflate(iset, "chrA", GRanges("chrB", IRanges(1, 10)))
```

```
y <- inflate(iset, 1:10, 1:10)
iset2 <- deflate(y)
iset2
assay(iset2)

y <- inflate(iset, 1:10, 1:10, swap=FALSE)
iset2 <- deflate(y)
iset2
assay(iset2)

# Testing with different fillings:
y <- inflate(iset, 1:10, 1:10, sample=2)
iset2 <- deflate(y)
assay(iset2)

y <- inflate(iset, 1:10, 1:10, fill=rowSums(assay(iset)))
iset2 <- deflate(y)
assay(iset2)

y2 <- inflate(interactions(iset), 1:10, 1:10, rowSums(assay(iset)))
identical(y, y2) # should be TRUE

# Effect of 'collapse'
y <- inflate(iset, c(8, 1:10), 1:10)
deflate(y)
deflate(y, collapse=FALSE)
```

`GInteractions class` *GInteractions class and constructors*

Description

The GInteractions class stores pairwise genomic interactions, and is intended for use in data analysis from Hi-C or ChIA-PET experiments. Each row of the GInteractions corresponds to a pairwise interaction between two loci, with indexing to improve computational efficiency.

Usage

```
## S4 method for signature 'numeric,numeric,GRanges'
GInteractions(anchor1, anchor2, regions, metadata=list(), mode="normal", ...)

## S4 method for signature 'GRanges,GRanges,GenomicRanges_OR_missing'
GInteractions(anchor1, anchor2, regions, metadata=list(), mode="normal", ...)

## S4 method for signature 'missing,missing,GenomicRanges_OR_missing'
GInteractions(anchor1, anchor2, regions, metadata=list(), mode="normal", ...)
```

Arguments

anchor1, anchor2	Either a pair of numeric vectors containing indices to regions, or a pair of GRanges objects specifying the interacting loci. Lengths of both anchor1 and anchor2 must be equal.
regions	A GRanges object containing the coordinates of the interacting regions. This is only mandatory if anchor1 and anchor2 are numeric vectors.
metadata	An optional list of arbitrary content describing the overall experiment.
mode	A string indicating what type of GInteractions object should be constructed.
...	Optional metadata columns.

Value

For the constructors, a GInteractions (or StrictGInteractions, or ReverseStrictGInteractions) object is returned.

Description of the class

The GInteractions class inherits from the Vector class and has access to all of its data members and methods (e.g, metadata and elementMetadata - see Vector-class for more details). It also contains several additional slots:

anchor1: An integer vector specifying the index of the first interacting region.

anchor2: An integer vector specifying the index of the second interacting region.

regions: A sorted GRanges object containing the coordinates of all interacting regions.

Each interaction is defined by the corresponding entries in the anchor1 and anchor2 slots, which point to the relevant coordinates in regions for each locus.

The StrictGInteractions class inherits from the GInteractions class and has the same slots. The only difference is that, for each interaction, anchor1 must be less than or equal to anchor2. This means that the first interacting region has a start position that is "lower" than the second interacting region. This condition is useful for comparing interactions within and between objects, as it ensures that redundant permutations of the same interaction are not being overlooked. However, it is not used by default as there may conceivably be instances where the order of interactions is informative. The ReverseStrictGInteractions class has the opposite behaviour, where anchor1 must be greater than or equal to anchor2.

Class construction

GInteractions objects can be constructed by specifying integer vectors to define the pairwise interactions in the GInteractions call. For entry x, the corresponding interaction is defined between regions[anchor1[x]] and regions[anchor2[x]]. Obviously, coordinates of all of the interacting loci must be specified in the regions argument. Any metadata in regions will be preserved. Note that regions will be resorted in the returned object, so the anchor indices may not be equal to the input anchor1 and anchor2.

Alternatively, GInteractions objects can be constructed by directly supplying the GRanges of the interacting loci to the GInteractions function. If regions is not specified, this will be constructed

automatically from the two sets of supplied GRanges. If `regions` is supplied, exact matching will be performed to identify the indices in `regions` corresponding to the regions in the supplied anchor GRanges. Missing values are not tolerated and will cause an error to be raised. In both cases, any metadata in the input GRanges will be transferred to the `mcols` of the output GInteractions object.

All constructors will return a GInteractions object containing all of the specified information. Sorting of `regions` is performed automatically, with re-indexing of all anchor indices to preserve the correct pairings between regions. If `mode="strict"`, a StrictGInterctions object is returned with anchor indices swapped such that `anchor1 <= anchor2` for all interactions. If `mode="reverse"`, a ReverseStrictGInterctions object is returned with anchor indices swapped such that `anchor1 >= anchor2`. If both anchors are missing, the constructor will return an empty GInteractions object.

Author(s)

Aaron Lun, with contributions from Malcolm Perry and Liz Ing-Simmons.

See Also

interaction-access, interaction-subset, interaction-compare, Vector-class

Examples

```
set.seed(1000)
N <- 30
all.starts <- sample(1000, N)
all.ends <- all.starts + round(runif(N, 5, 20))
all.regions <- GRanges(rep(c("chrA", "chrB"), c(N-10, 10)),
    IRanges(all.starts, all.ends))

Np <- 20
all.anchor1 <- sample(N, Np)
all.anchor2 <- sample(N, Np)
gi <- GInteractions(all.anchor1, all.anchor2, all.regions)

# Equivalent construction:
GInteractions(all.regions[all.anchor1], all.regions[all.anchor2])
GInteractions(all.regions[all.anchor1], all.regions[all.anchor2],
    all.regions)

# Putting in metadata, elementMetadata
temp.gi <- gi
metadata(temp.gi)$name <- "My GI object"
mcols(temp.gi)$score <- runif(Np)

# Strict construction
sgi <- GInteractions(all.regions[all.anchor1], all.regions[all.anchor2],
        all.regions, mode="strict")
rsgi <- GInteractions(all.regions[all.anchor1], all.regions[all.anchor2],
        all.regions, mode="reverse")
```

GRanges methods *Equivalents of GRanges methods*

Description

Methods for GInteractions, InteractionSet and ContactMatrix that operate on the internal GenomicRanges.

Usage

```
## S4 method for signature 'GInteractions'
shift(x, shift=0L, use.names=TRUE)
## S4 method for signature 'InteractionSet'
shift(x, shift=0L, use.names=TRUE)
## S4 method for signature 'ContactMatrix'
shift(x, shift=0L, use.names=TRUE)

## S4 method for signature 'GInteractions'
narrow(x, start=NA, end=NA, width=NA, use.names=TRUE)
## S4 method for signature 'InteractionSet'
narrow(x, start=NA, end=NA, width=NA, use.names=TRUE)
## S4 method for signature 'ContactMatrix'
narrow(x, start=NA, end=NA, width=NA, use.names=TRUE)

## S4 method for signature 'GInteractions'
resize(x, width, fix="start", use.names=TRUE, ...)
## S4 method for signature 'InteractionSet'
resize(x, width, fix="start", use.names=TRUE, ...)
## S4 method for signature 'ContactMatrix'
resize(x, width, fix="start", use.names=TRUE, ...)

## S4 method for signature 'GInteractions'
flank(x, width, start=TRUE, both=FALSE, use.names=TRUE, ignore.strand=FALSE)
## S4 method for signature 'InteractionSet'
flank(x, width, start=TRUE, both=FALSE, use.names=TRUE, ignore.strand=FALSE)
## S4 method for signature 'ContactMatrix'
flank(x, width, start=TRUE, both=FALSE, use.names=TRUE, ignore.strand=FALSE)

## S4 method for signature 'GInteractions'
trim(x, use.names=FALSE)
## S4 method for signature 'InteractionSet'
trim(x, use.names=FALSE)
## S4 method for signature 'ContactMatrix'
trim(x, use.names=FALSE)

## S4 method for signature 'GInteractions'
width(x)
```

```
## S4 method for signature 'InteractionSet'
width(x)
## S4 method for signature 'ContactMatrix'
width(x)
```

Arguments

x	A GInteractions, InteractionSet or ContactMatrix object.
shift, start, end, width, fix, both	Further interaction-specific arguments to pass to the relevant GenomicRanges methods, see details.
use.names, ignore.strand, ...	Further overall arguments to pass to the relevant GenomicRanges methods.

Details

The trim, resize, narrow and shift methods will apply the GenomicRanges methods with the same name to the anchor regions of x. For example, resize(x, width) will produce an object that is equivalent to running resize on the first and second anchor regions directly. This is *not* the same as running the methods on the regions slot of x, which is an important distinction when the arguments are vectors.

The interaction-specific arguments can be scalars, vectors or a list of two scalars/vectors. Scalars and vectors will be recycled across the length of the first and second anchor regions. For lists, the first element will be applied to the first anchor regions, and the second element will be applied to the second anchor regions.

After any modifications are applied, resorting may be performed to ensure that the entries in the regions slot of the output object are ordered. This means that the order of the ranges in the regions slot may change between the input and output object. However, the number and order of the *interactions* will not change.

For GInteractions and InteractionSet objects, the width method will return a list with two integer vectors first and second with the same length as x. These contain the widths of the first or second anchor regions corresponding to each interaction. For ContactMatrix objects, the method will return a list with the vectors row and column, with lengths equal to the number of rows and column respectively.

Value

Depending on the method, an object of the same class as x, or information regarding the genomic regions – see Details.

Author(s)

Aaron Lun, based on suggestions from Liz Ing-Simmons.

See Also

trim,GenomicRanges-method, resize,GenomicRanges-method, narrow,GenomicRanges-method, shift,GenomicRanges-method, flank,GenomicRanges-method, width,IRanges-method

Examples

```
example(GInteractions, echo=FALSE)

trim(gi)
width(gi)

# Functions are applied along the length of 'gi':
new.sizes <- round(runif(length(gi), 10, 50))
gi2 <- resize(gi, width=new.sizes)
gi2
resize(first(gi), width=new.sizes)
resize(second(gi), width=new.sizes)

# ... not along the length of 'regions(gi)': note the difference!
mod.regions <- suppressWarnings(resize(regions(gi), width=new.sizes))
mod.regions[anchors(gi, type="first", id=TRUE)] # not the same as first(gi2)!
mod.regions[anchors(gi, type="second", id=TRUE)] # not the same second(gi2)!

example(ContactMatrix, echo=FALSE)
trim(x)
width(x)
```

Interaction accessors *Interaction accessors*

Description

Methods to get and set fields in an InteractionSet or GInteractions object.

Usage

```
## S4 method for signature 'GInteractions'
anchors(x, type="both", id=FALSE)
## S4 method for signature 'InteractionSet'
anchors(x, type="both", id=FALSE)

## S4 method for signature 'GInteractions'
anchorIds(x, type="both")
## S4 method for signature 'InteractionSet'
anchorIds(x, type="both")

## S4 method for signature 'GInteractions'
first(x)
## S4 method for signature 'InteractionSet'
first(x)

## S4 method for signature 'GInteractions'
second(x)
```

```
## S4 method for signature 'InteractionSet'
second(x)

## S4 method for signature 'GInteractions'
regions(x)
## S4 method for signature 'InteractionSet'
regions(x)

## S4 method for signature 'GInteractions'
seqinfo(x)
## S4 method for signature 'InteractionSet'
seqinfo(x)

## S4 method for signature 'GInteractions'
show(x)
## S4 method for signature 'InteractionSet'
show(x)

## S4 replacement method for signature 'GInteractions'
anchorIds(x, type="both") <- value
## S4 replacement method for signature 'InteractionSet'
anchorIds(x, type="both") <- value

## S4 replacement method for signature 'GInteractions'
regions(x) <- value
## S4 replacement method for signature 'InteractionSet'
regions(x) <- value

## S4 replacement method for signature 'GInteractions'
seqinfo(x, new2old = NULL,
    pruning.mode = c("error", "coarse", "fine", "tidy")) <- value
## S4 replacement method for signature 'InteractionSet'
seqinfo(x, new2old = NULL,
    pruning.mode = c("error", "coarse", "fine", "tidy")) <- value

## S4 replacement method for signature 'GInteractions'
replaceRegions(x) <- value
## S4 replacement method for signature 'InteractionSet'
replaceRegions(x) <- value

## S4 replacement method for signature 'GInteractions'
appendRegions(x) <- value
## S4 replacement method for signature 'InteractionSet'
appendRegions(x) <- value

## S4 method for signature 'GInteractions'
reduceRegions(x)
## S4 method for signature 'InteractionSet'
```

```
reduceRegions(x)

## S4 method for signature 'GInteractions'
names(x)
## S4 method for signature 'InteractionSet'
names(x)

## S4 replacement method for signature 'GInteractions'
names(x) <- value
## S4 replacement method for signature 'InteractionSet'
names(x) <- value

## S4 replacement method for signature 'StrictGInteractions'
anchors(x, type="both") <- value
## S4 replacement method for signature 'ReverseStrictGInteractions'
anchors(x, type="both") <- value

## S4 method for signature 'GInteractions'
length(x)
## S4 method for signature 'GInteractions'
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
## S4 method for signature 'GInteractions'
x$name
## S4 replacement method for signature 'GInteractions'
x$name <- value

## S4 method for signature 'InteractionSet'
interactions(x)
## S4 replacement method for signature 'InteractionSet'
interactions(x) <- value

## S4 method for signature 'InteractionSet'
mcols(x, use.names=FALSE)
## S4 replacement method for signature 'InteractionSet'
mcols(x) <- value
```

Arguments

x	An InteractionSet or GInteractions object.
type	a string specifying which anchors are to be extracted or replaced.
id	a scalar logical indicating whether indices or GRanges should be returned.
new2old, pruning.mode	Additional arguments to pass to seqinfo,GRanges-method.
name	a string indicating the field of mcols to be accessed or modified for a GInteractions object.
value	For anchorIds<- and type="first" or "second", an integer vector of length equal to the number of rows in x. For type="both", a list of two such vectors must be supplied.

For regions<-, a GRanges object of length equal to that of regions(x). For replaceRegions<-, a GRanges object that is a superset of all entries in regions(x) involved in interactions. For appendRegions<-, a GRanges of any length containing additional regions.

For seqinfo<-, a SeqInfo object containing chromosome length data for all regions. For interactions<-, a GInteractions object of length equal to that of interactions(x). For mcols<-, a DataFrame with number of rows equal to the length of x. For names<-, a character vector of length equal to that of x.

row.names, optional, ...

Additional arguments, see ?as.data.frame for more details.

use.names A logical scalar, see ?mcols for more details.

Value

For the getters, values in various slots of x are returned, while for the setters, the slots of x are modified accordingly – see Details.

Anchor manipulations for GInteractions

The return value of anchors varies depending on type and id:

- If id=FALSE, a GRanges object is returned containing the regions specified by the anchor1 or anchor2 slots in x, for type=="first" or "second", respectively. The first and second methods are synonyms for anchors in these respective cases.
- If id=FALSE and type="both", a list is returned with two entries first and second, containing regions specified by anchor1 and anchor2 respectively.
- If id=TRUE and type="both", a list is returned containing the two integer vectors in the anchor1 or anchor2 slots of x. Otherwise, each vector is returned corresponding to the requested value of type.

Note that anchorIds is the same as calling anchors with id=TRUE.

Replacement in anchorIds<- requires anchor indices rather than a GRanges object. If type="both", a list of two integer vectors is required in value, specifying the indices of the interacting regions in regions(x). If type="first" or "second", an integer vector is required to replace the existing values in the anchor1 or anchor2 slot, respectively. If the object is a StrictGInteractions, indices are automatically swapped so that anchor1 >= anchor2 for each interaction. The opposite applies if the object is a ReverseStrictGInteractions.

Region manipulations for GInteractions

For regions, a GRanges is returned equal to the regions slot in x. For regions<-, the GRanges in value is used to replace the regions slot. Resorting of the replacement GRanges is performed automatically, along with re-indexing of the anchors. In addition, the replacement must be of the same length as the existing object in the regions slot.

The replaceRegions replacement method can take variable length GRanges, but requires that the replacement contain (at least) all ranges contained in anchors(x). The appendRegions replacement method appends extra intervals to the existing regions slot of x. This is more efficient than calling replaceRegions on a concatenated object with regions(x) and the extra intervals.

The reduceRegions method will return a GInteractions object where the regions slot is reduced to only those entries used in anchors(x). This may save some memory in each object by removing unused regions. However, this is not recommended for large workflows with many GInteractions objects. R uses a copy-on-write memory management scheme, so only one copy of the GRanges in regions should be stored so long as it is not modified in different objects.

Other methods for GInteractions

For access and setting of all other slots, see Vector-class for details on the appropriate methods. This includes mcols or metadata, to store interactions-specific or experiment-wide metadata, respectively. The length method will return the number of interactions stored in x.

The show method will print out the class, the number of pairwise interactions, and the total number of regions in the GInteractions object. The number and names of the various metadata fields will also be printed. The as.data.frame method will return a data.frame object containing the coordinates for the two anchor regions as well as any element-wise metadata.

The seqinfo method will return the sequence information of the GRanges in the regions slot. This can be replaced with the seqinfo<- method – see ?Seqinfo for more details.

Details for InteractionSet

Almost all InteractionSet methods operate by calling the equivalent method for the GInteractions object, and returning the resulting value. The only exception is interactions, which returns the GInteractions object in the interactions slot of the InteractionSet object. This slot can also be set by supplying a valid GInteractions object in interactions<-.

The show method will print information equivalent to that done for a SummarizedExperiment object. An additional line is added indicating the number of regions in the regions slot of the object.

For access and setting of all other slots, see SummarizedExperiment-class for details on the appropriate methods. This includes assays, colData, mcols or metadata, which can all be applied to InteractionSet objects.

Handling different metadata

Note that there are several options for metadata - experiment-wide metadata, which goes into metadata(x)<-; region-specific metadata (e.g., adjacent genes, promoter/enhancer identity, GC content), which goes into mcols(regions(x))<-; and interaction-specific metadata (e.g., interaction distance, interaction type), which goes directly into mcols(x)<-. This is applicable to both GInteractions and InteractionSet objects. In addition, library-specific data (e.g., library size) should be placed into colData(x)<- for InteractionSet objects.

Users should take care with the differences in the $ and $<- operators between these two classes. For GInteractions objects, this will access or modify fields in the mcols slot, i.e., for interaction-specific metadata. For InteractionSet objects, this will access or modify fields in the colData slot, i.e., for library-specific data. The difference in behaviour is due to the concept of libraries in the InteractionSet, which is lacking in the GInteractions class.

For InteractionSet objects, the mcols getter and setter functions operate on the GInteractions object stored in interactions slot, rather than accessing the elementMetadata slot of the SummarizedExperiment base class. This makes no difference for practical usage in the vast majority of cases, except that any metadata stored in this manner is carried over when the GInteractions object is

extracted with `interactions(x)`. Similarly, the `names` getter and setter will operate the names of the internal GInteractions object. However, the `metadata` getter and setter will operate on the SummarizedExperiment base class, *not* on the internal GInteractions object.

Author(s)

Aaron Lun

See Also

GInteractions-class, InteractionSet-class, Vector-class, SummarizedExperiment-class

Examples

```
example(GInteractions, echo=FALSE) # Generate a nice object.
show(gi)

# Accessing anchor ranges or indices:
anchors(gi)
anchors(gi, type="first")
anchors(gi, id=TRUE)

anchors(gi, id=TRUE, type="first")
anchors(gi, id=TRUE, type="second")

# Modifying anchor indices:
nregs <- length(regions(gi))
mod <- list(sample(nregs, length(gi), replace=TRUE),
            sample(nregs, length(gi), replace=TRUE))
anchorIds(gi) <- mod
anchors(gi, id=TRUE, type="first")
anchors(gi, id=TRUE, type="second")

anchorIds(gi, type="both") <- mod
anchorIds(gi, type="first") <- mod[[1]]
anchorIds(gi, type="first") <- mod[[2]]

# Accessing or modifying regions:
regions(gi)
reduceRegions(gi)
regions(gi)$score <- runif(length(regions(gi)))

new.ranges <- c(regions(gi), resize(regions(gi), fix="center", width=50))
try(regions(gi) <- new.ranges) # Fails
replaceRegions(gi) <- new.ranges # Succeeds

length(regions(gi))
appendRegions(gi) <- GRanges("chrA", IRanges(5:10+1000, 1100+5:10), score=runif(6))
length(regions(gi))

seqinfo(gi)
seqinfo(gi) <- Seqinfo(seqnames=c("chrA", "chrB"), seqlengths=c(1000, 2000))
```

```
# Accessing or modifying metadata.
gi$score <- runif(length(gi))
mcols(gi)
as.data.frame(gi)

#################
# Same can be done for an InteractionSet object:

example(InteractionSet, echo=FALSE)

anchors(iset)
regions(iset)
reduceRegions(iset)
regions(iset)$score <- regions(gi)$score
replaceRegions(iset) <- new.ranges

seqinfo(iset)
seqinfo(iset) <- Seqinfo(seqnames=c("chrA", "chrB"), seqlengths=c(1000, 2000))

# Standard SE methods also available:
colData(iset)
metadata(iset)
mcols(iset)

# Note the differences in metadata storage:
metadata(iset)$name <- "metadata for SE0"
metadata(interactions(iset))$name <- "metadata for GI"

iset$lib.size <- runif(ncol(iset))*1e6
colData(iset)
mcols(iset) # untouched by "$" operator

mcols(iset)$whee <- runif(nrow(iset))
mcols(iset)
mcols(interactions(iset)) # preserved

names(iset) <- paste0("X", seq_along(iset))
names(iset)
names(interactions(iset))
```

`Interaction binding` *Interaction combining*

Description

Methods to combine InteractionSet or GInteractions objects.

Usage

```
## S4 method for signature 'GInteractions'
c(x, ..., recursive=FALSE)

## S4 method for signature 'InteractionSet'
rbind(..., deparse.level=1)
## S4 method for signature 'InteractionSet'
cbind(..., deparse.level=1)
```

Arguments

x	A GInteractions or InteractionSet object.
...	For rbind and c, ... contains GInteractions or InteractionSet objects to be combined row-wise. All objects must be of the same class. For c, any objects are additional to that already specified in x. For cbind, ... contains InteractionSet objects to be combined column-wise.
deparse.level	An integer scalar; see ?base::cbind for a description of this argument.
recursive	An integer scalar, ignored.

Value

A combined object of the same class as x.

Details for GInteractions

c will concatenate GInteractions objects. It will check whether the regions slot of all supplied objects are the same, in which case the regions and anchor indices are used directly. Otherwise, the regions slot is set to a new GRanges object consisting of the (sorted) union of all regions across the input objects. Anchor indices in each object are refactored appropriately to refer to the relevant entries in the new GRanges.

Note that the column names in mcols must be identical across all supplied objects. The column names of mcols for the regions slot must also be identical across objects. If GInteractions objects of different strictness (i.e., StrictGInteractions and ReverseGInteractions) are concatenated, the returned object will be of the same class as the first supplied object.

Details for InteractionSet

cbind will combine objects with the same interactions but different samples. Errors will occur if the interactions are not identical between objects (i.e., must have same values in the interactions slots). Additional restrictions apply on the column and assay names - see cbind,SummarizedExperiment-method for details.

rbind will combine objects with the same samples but different interactions. In this case, the interactions need not be identical, and will be concatenated using the methods described above for GInteractions objects. Again, additional restrictions apply - see rbind,SummarizedExperiment-method for details.

Author(s)

Aaron Lun

See Also

InteractionSet-class GInteractions-class

Examples

```
example(GInteractions, echo=FALSE)
c(gi, gi)

new.gi <- gi
regions(new.gi) <- resize(regions(new.gi), width=20, fix="start")
c(gi, new.gi)

#################
# Same can be done for an InteractionSet object:

example(InteractionSet, echo=FALSE)
cbind(iset, iset)
rbind(iset, iset)

new.iset <- iset
regions(new.iset) <- resize(regions(new.iset), width=20, fix="start")
rbind(iset, new.iset)
```

Interaction compare *Interaction comparisons*

Description

Methods to order, compare and de-duplicate GInteractions or InteractionSet objects, based on the anchor indices.

Usage

```
## S4 method for signature 'GInteractions'
order(..., na.last=TRUE, decreasing=FALSE)

## S4 method for signature 'GInteractions'
sort(x, decreasing=FALSE, ...)

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

## S4 method for signature 'GInteractions'
unique(x, incomparables=FALSE, fromLast=FALSE, ...)
```

```
## S4 method for signature 'GInteractions'
swapAnchors(x, mode=c("order", "reverse", "all"))

## Each of the above methods has an identical equivalent for
## InteractionSet objects (not shown for brevity).

## S4 method for signature 'GInteractions,GInteractions'
match(x, table, nomatch=NA_integer_, incomparables=NULL, ...)

## S4 method for signature 'GInteractions,InteractionSet'
match(x, table, nomatch=NA_integer_, incomparables=NULL, ...)

## S4 method for signature 'InteractionSet,GInteractions'
match(x, table, nomatch=NA_integer_, incomparables=NULL, ...)

## S4 method for signature 'InteractionSet,InteractionSet'
match(x, table, nomatch=NA_integer_, incomparables=NULL, ...)

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

Arguments

...	For order, one or more InteractionSet or GInteractions objects with the same number of rows. Otherwise, ignored in all other methods.
x	An InteractionSet or GInteractions object. For pcompare, a GInteractions object only.
na.last	A logical scalar indicating whether NA values should be ordered last. This should not be relevant as anchor indices should be finite.
decreasing	A logical scalar indicating whether rows should be sorted in decreasing order.
incomparables	A logical scalar. See ?match for a description of this in match. Otherwise, it is ignored.
fromLast	A logical scalar indicating whether the last entry of a repeated set in x should be considered as a non-duplicate.
mode	A string indicating what type of swapping should be performed in swapAnchors.
table	A GInteractions or InteractionSet object.
nomatch	An integer scalar indicating the value to use for unmatched entries.
y	A GInteractions object, of the same length as x.

Value

For sort and unique, a GInteractions or InteractionSet object is returned, depending on the class of x.

For order and duplicated, an integer vector of permutations or a logical vector of duplicate specifications is returned, respectively.

Sorting and ordering

Sorting is performed based on the anchor indices of the GInteraction object. Rows are ordered for increasing values of the anchor1 slot - if these are equal, ordering is performed with values of the anchor2 slot. This equates to ordering by the coordinates directly, as the GRanges in the regions slot is always sorted. Based on this, sort will return a sorted copy of x with permuted rows in increasing order.

The order method returns an integer vector indicating the permutation to rearrange x in increasing order. If multiple objects are supplied to order, ordering will be computed using anchor indices from successive objects. For example, ordering is first performed using anchor indices from the first object; any rows with the same anchor1 and anchor2 will be split using the second object; and so on.

For both of these methods, the sorting can be reversed by setting decreasing=TRUE. This will sort or order for decreasing values of anchor1 and anchor2, rather than for increasing values.

Removing duplicates

The duplicated method returns a logical vector indicating whether the rows of x are duplicated. Duplicated rows are identified on the basis of identical entries in the anchor1 and anchor2 slots. Values in other slots (e.g., in mcols) are ignored. For a set of duplicated rows, the first occurrence in x is marked as the non-duplicate if fromLast=FALSE, and the last entry otherwise.

unique returns a GInteractions object where all duplicate rows have been removed from x. This is equivalent to x[!duplicated(x),], with any additional arguments passed to duplicated as specified.

Matching and comparing

The match function will return an integer vector of length equal to that of x. Each entry of the vector corresponds to an interaction in x and contains the index of the first interaction table with the same anchor regions. Interactions in x without any matches are assigned NA values by default. If the regions slot is not the same between x and table, match will call findOverlaps to identify exact two-dimensional overlaps.

The pcompare function will return an integer vector of length equal to x and y. This performs parallel comparisons between corresponding entries in two GInteractions objects, based on the values of the anchor indices (anchor1 first, and then anchor2 if anchor1 is tied). Negative, zero and positive values indicate that the corresponding interaction in x is "lesser", equal or "greater" than the corresponding interaction in y. If the regions slot is not the same between the two objects, the union of regions for both objects will be used to obtain comparable indices.

Swapping anchors

For GInteractions objects, swapAnchors returns a GInteractions object where anchor1 and anchor2 values are swapped. If mode="order", this is done so that all values in the anchor2 slot are not less than values in anchor1 in the returned object. If mode="reverse", all values in anchor1 are not less than all values in anchor2. If mode="all", the anchor indices in x are directly swapped without consideration of ordering.

It is recommended to apply this method before running methods like order or duplicated. This ensures that redundant permutations are coerced into the same format for a valid comparison. In

many applications, permutations of pairwise interactions are not of interest, i.e., an interaction between regions 1 and 2 is the same as that between 2 and 1. Application of `swapAnchors` with `mode="order"` ensures that all indices are arranged in a comparable manner. Alternatively, users can use a (Reverse)StrictGInteractions object which enforces a consistent arrangement of indices across interactions.

Methods for InteractionSet objects

For all InteractionSet methods, the corresponding method is called on the GInteractions object in the `interactions` slot of the InteractionSet object. Return values for each InteractionSet method is the same as those for the corresponding GInteractions method - except for `sort` and `unique`, which return a row-permuted or row-subsetted InteractionSet, respectively, instead of a GInteractions object; and `swapAnchors`, which returns an InteractionSet object where the internal GInteractions has its anchor indices swapped around.

Note that no additional information from the InteractionSet (beyond that in `interactions`) is used for sorting or duplicate marking, i.e., the assay or metadata values for each interaction are *not* used in distinguishing rows with the same interaction. For this reason, the `pcompare` method is not implemented for InteractionSet objects. It makes little sense to do a parallel comparison in an InteractionSet without examining the data.

Author(s)

Aaron Lun

See Also

GInteractions-class, InteractionSet-class, match, pcompare

Examples

```
example(GInteractions, echo=FALSE)

anchors(gi, id=TRUE)
anchors(swapAnchors(gi, mode="all"), id=TRUE)
gi <- swapAnchors(gi)

anchors(gi)
gi2 <- sort(gi)
gi2
anchors(gi2)

# Can also order them.
o <- order(gi)
o
stopifnot(all.equal(gi[o], gi2))

# Checking duplication.
summary(duplicated(gi))
summary(duplicated(c(gi, gi)))
stopifnot(all.equal(unique(gi), unique(c(gi, gi))))
```

```
# Matching and comparing.
another.gi <- gi[sample(length(gi))]
match(gi, another.gi)
match(gi, another.gi[1:5])

pcompare(gi, another.gi)

#################
# Same can be done for an InteractionSet object:

example(InteractionSet, echo=FALSE)
iset <- swapAnchors(iset)

anchors(iset)
anchors(sort(iset))
order(iset)
summary(duplicated(iset))
unique(iset)
```

`Interaction distances` *Get the linear distance for each interaction*

Description

Compute the distance between interacting regions on the linear genome, for each pairwise interaction contained in a GInteractions or InteractionSet object.

Usage

```
## S4 method for signature 'GInteractions'
pairdist(x, type="mid")
## S4 method for signature 'InteractionSet'
pairdist(x, type="mid")

## S4 method for signature 'GInteractions'
intrachr(x)
## S4 method for signature 'InteractionSet'
intrachr(x)
```

Arguments

`x`	A GInteractions or InteractionSet object.
`type`	A character string specifying the type of distance to compute. Can take values of `"mid"`, `"gap"`, `"span"`, `"diag"` or `"intra"`.

Details

For each interaction in x, the pairdist method computes the distance between the two interacting regions. An integer vector is returned, with values computed according to the specified value of type:

"mid": The distance between the midpoints of the two regions (rounded down to the nearest integer) is returned.

"gap": The length of the gap between the closest points of the two regions is computed - negative lengths are returned for overlapping regions, indicating the length of the overlap.

"span": The distance between the furthermost points of the two regions is computed.

"diag": The difference between the anchor indices is returned. This corresponds to a diagonal on the interaction space when bins are used in the regions slot of x.

Interchromosomal interactions are marked with NA. Alternatively, if type="intra", a logical vector is returned indicating whether the interaction occurs between two regions on the same chromosome. intrachr(x) is an alias for pairdist(x, type="intra").

The return values of the assorted methods are the same for both GInteractions and InteractionSet objects. Methods for the latter operate on the GInteractions object in the interactions slot.

Value

An integer or logical vector of distances.

Author(s)

Aaron Lun

See Also

GInteractions-class, InteractionSet-class

Examples

```
example(GInteractions, echo=FALSE)

pairdist(gi)
pairdist(gi, type="gap")
pairdist(gi, type="span")
pairdist(gi, type="diag")
intrachr(gi)

example(InteractionSet, echo=FALSE)

pairdist(iset)
pairdist(iset, type="gap")
pairdist(iset, type="span")
pairdist(iset, type="diag")
intrachr(iset)
```

`Interaction overlaps` *Find overlaps between interactions in one or two dimensions*

Description

Find overlaps between interactions and linear intervals, between interactions and pairs of intervals, and between interactions and other interactions in a GInteractions or InteractionSet object.

Usage

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

## S4 method for signature 'GInteractions,GInteractions'
overlapsAny(query, subject, maxgap=-1L, minoverlap=0L,
    type=c("any", "start", "end", "within", "equal"),
    ignore.strand=TRUE, ..., use.region="both")

## S4 method for signature 'GInteractions,GInteractions'
countOverlaps(query, subject, maxgap=-1L, minoverlap=0L,
    type=c("any", "start", "end", "within", "equal"),
    ignore.strand=TRUE, ..., use.region="both")

## S4 method for signature 'GInteractions,GInteractions'
subsetByOverlaps(query, subject, maxgap=-1L, minoverlap=0L,
    type=c("any", "start", "end", "within", "equal"),
    ignore.strand=TRUE, ..., use.region="both")
```

Arguments

`query`, `subject`	A Vector, GInteractions or InteractionSet object, depending on the specified method. At least one of these must be a GInteractions or InteractionSet object. Also, `subject` can be missing if `query` is a GInteractions or InteractionSet object.
`maxgap`, `minoverlap`, `type`, `select`	See ?`findOverlaps` in the **GenomicRanges** package.
`ignore.strand`	A logical scalar indicating whether strand information in `query` or `subject` should be ignored. Note that the default setting here is different to that in `findOverlaps` as genomic interactions are usually unstranded.
`...`	Further arguments to pass to `findOverlaps`. This includes `drop.self` and `drop.redundant` when `subject` is missing.
`use.region`	A string specifying the regions to be used to identify overlaps.

Value

For `findOverlaps`, a Hits object is returned if `select="all"`, and an integer vector of subject indices otherwise.

For `countOverlaps` and `overlapsAny`, an integer or logical vector is returned, respectively.

For `subsetByOverlaps`, a subsetted object of the same class as `query` is returned.

Overview of overlaps for GInteractions

All methods can be applied using a GInteractions as either the `query` or `subject`, and a Vector as the other argument. In such cases, the Vector is assumed to represent some region on the linear genome (e.g., GRanges) or set of such regions (GRangesList). An overlap will be defined between the interval and an GInteractions interaction if either anchor region of the latter overlaps the former. This is considered to be a one-dimensional overlap, i.e., on the linear genome.

The same methods can be applied using two GInteractions objects as the `query` and `subject`. In such cases, a two-dimensional overlap will be computed between the anchor regions of the two objects. An overlap is defined if each anchor region of the first object overlaps at least one anchor region of the second object, and each anchor region of the second object overlaps at least one anchor region of the first object, i.e., there are overlapping areas in the two-dimensional interaction space. If `subject` is missing, overlaps will be computed between interactions in `query`.

Description of overlap methods

When `select="all"`, `findOverlaps` returns a Hits object containing overlapping pairs of queries and subjects (or more specifically, their indices in the supplied objects - see `?findOverlaps` for more details). For other values of `select`, an integer vector is returned with one entry for each element of `query`, which specifies the index of the chosen (first, last or arbitrary) overlapping feature in `subject` for that query. Queries with no overlaps at all are assigned `NA` values.

For the other methods, `countOverlaps` returns an integer vector indicating the number of elements in `subject` that were overlapped by each element in `query`. `overlapsAny` returns a logical vector indicating which elements in `query` were overlapped by at least one element in `subject`. `subsetByOverlaps` returns a subsetted `query` containing only those elements overlapped by at least one element in `subject`.

Choice of regions to define overlaps

For one-dimensional overlaps, `use.region="both"` by default such that overlaps with either anchor region are considered. If `use.region="first"`, overlaps are only considered between the interval and the first anchor region. Similarly, if `use.region="second"`, only the second anchor region is used.

Equivalent choices are available for two-dimensional overlaps:

- By default, `use.region="both"` such that the order of first/second anchor regions in the query and subject is ignored. This means that the first anchor region in the query can overlap both the first *or* second anchor regions in the subject. Similarly, the second anchor region in the query can overlap both the first *or* ssecond anchor regions in the subject.
- If `use.region="same"`, overlaps are only considered between the first anchor regions for the query and the subject, or between the second anchor regions for the query and subject.

Overlaps between the first query region and the second subject region, or the second query region and the first subject region, are ignored.

- If `use.region="reverse"`, overlaps are only considered between the first anchor regions for the query and the second anchor regions for the subject, and vice versa. Overlaps between the first query/subject regions or between the second query/subject regions are ignored.

The latter two options tend only to be useful if the order of first/second regions is informative.

Details for InteractionSet

Each method can also be applied with InteractionSet objects, and the behaviour is largely the same as that described for GInteractions objects. For a given InteractionSet object `x`, the corresponding method is called on the GInteractions object in the `interactions` slot of `x`. The return value is identical to that from calling the method on `interactions(x)`, except for `subsetByOverlaps` for InteractionSet queries (which returns a subsetted InteractionSet object, containing only those rows/interactions overlapping the `subject`).

Author(s)

Aaron Lun

See Also

InteractionSet-class, findOverlaps, linkOverlaps

Examples

```
example(GInteractions, echo=FALSE)

# Making a larger object, for more overlaps.
Np <- 100
N <- length(regions(gi))
all.anchor1 <- sample(N, Np, replace=TRUE)
all.anchor2 <- sample(N, Np, replace=TRUE)
gi <- GInteractions(all.anchor1, all.anchor2, regions(gi))

# GRanges overlaps:
of.interest <- resize(sample(regions(gi), 2), width=1, fix="center")
findOverlaps(of.interest, gi)
findOverlaps(gi, of.interest)
findOverlaps(gi, of.interest, select="first")
overlapsAny(gi, of.interest)
overlapsAny(of.interest, gi)
countOverlaps(gi, of.interest)
countOverlaps(of.interest, gi)
subsetByOverlaps(gi, of.interest)
subsetByOverlaps(of.interest, gi)

# GRangesList overlaps:
pairing <- GRangesList(first=regions(gi)[1:3], second=regions(gi)[4:6],
    third=regions(gi)[7:10], fourth=regions(gi)[15:17])
```

```
findOverlaps(pairing, gi)
findOverlaps(gi, pairing)
findOverlaps(gi, pairing, select="last")
overlapsAny(gi, pairing)
overlapsAny(pairing, gi)
countOverlaps(gi, pairing)
countOverlaps(pairing, gi)
subsetByOverlaps(gi, pairing)
subsetByOverlaps(pairing, gi)

# GInteractions overlaps (split into two):
first.half <- gi[1:(Np/2)]
second.half <- gi[Np/2+1:(Np/2)]
findOverlaps(first.half, second.half)
findOverlaps(first.half, second.half, select="arbitrary")
overlapsAny(first.half, second.half)
countOverlaps(first.half, second.half)
subsetByOverlaps(first.half, second.half)

findOverlaps(gi)
countOverlaps(gi)
overlapsAny(gi) # trivial result

##################
# Same can be done for an InteractionSet object:

Nlibs <- 4
counts <- matrix(rpois(Nlibs*Np, lambda=10), ncol=Nlibs)
colnames(counts) <- seq_len(Nlibs)
iset <- InteractionSet(counts, gi)

findOverlaps(of.interest, iset)
findOverlaps(iset, pairing)
findOverlaps(iset[1:(Np/2),], iset[Np/2+1:(Np/2),])

# Obviously returns InteractionSet objects instead
subsetByOverlaps(of.interest, iset)
subsetByOverlaps(iset, pairing)
subsetByOverlaps(iset[1:(Np/2),], iset[Np/2+1:(Np/2),])

# Self-overlaps
findOverlaps(iset)
countOverlaps(iset)
overlapsAny(iset) # trivial result
```

Interaction subsetting

Interaction subsetting and combining

Description

Methods to subset or combine InteractionSet or GInteractions objects.

Usage

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

## S4 replacement method for signature 'InteractionSet,ANY,ANY,InteractionSet'
x[i, j] <- value

## S4 method for signature 'InteractionSet'
subset(x, i, j)

## S4 replacement method for signature 'GInteractions,ANY,GInteractions'
x[i] <- value
```

Arguments

`x`	A GInteractions or InteractionSet object.
`i, j`	A vector of logical or integer subscripts. For InteractionSet objects, these indicate the rows and columns to be subsetted for `i` and `j`, respectively. Rows correspond to pairwise interactions while columns correspond to samples. For GInteractions objects, `i` indicates the genomic interactions to be retained. `j` is ignored as there is no concept of samples in this class.
`..., drop`	Additional arguments that are ignored.
`value`	A GInteractions or InteractionSet object with length or number of rows equal to length of `i` (or that of `x`, if `i` is not specified). For InteractionSet objects, the number of columns must be equal to the length of `j` (or number of columns in `x`, if `j` is not specified).

Value

A subsetted object of the same class as `x`.

Details for GInteractions

Subsetting operations are not explicitly listed above as they inherit from the `Vector` class. They will return a GInteractions object containing the specified interactions. Values of the `anchor1` and `anchor2` slots will be appropriately subsetted in the returned object, along with any metadata in `mcols`. However, note that the value of `regions` will not be modified by subsetting.

For short index vectors, subsetting a GInteractions object prior to calling `anchors` may be much faster than the reverse procedure. This is because the `anchors` getter will construct a GRanges(List) containing the genomic loci for all pairwise interactions. Subsetting beforehand ensures that only loci for the desired interactions are included. This avoids constructing the entire object just to subset it later.

Subset assignment will check if the `regions` are identical between `x` and `value`. If not, the `regions` slot in the output object will be set to a sorted union of all `regions` from `x` and `value`. Indices are refactored appropriately to point to the entries in the new `regions`.

Details for InteractionSet

Subsetting behaves in much the same way as that for the SummarizedExperiment class. Interactions are treated as rows and will be subsetted as such. All subsetting operations will return an InteractionSet with the specified interactions (rows) or samples (columns). Again, note that the value of `regions` will not be modified by subsetting.

Author(s)

Aaron Lun

See Also

InteractionSet-class GInteractions-class

Examples

```
example(GInteractions, echo=FALSE)

# Subsetting:
gi[1,]
gi[1:2,]
gi[3]
gi[3:4]

temp.gi <- gi
temp.gi[3:4] <- gi[1:2]

# Splitting:
f <- sample(4, length(gi), replace=TRUE)
out <- split(gi, f)
out[[1]]

#################
# Same can be done for an InteractionSet object:

example(InteractionSet, echo=FALSE)

# Subsetting:
iset[1,]
iset[1:2,]
iset[,1]
iset[,1:2]
iset[3,3]
iset[3:4,3:4]

# Splitting:
out <- split(iset, f)
```

```
out[[1]]
```

`InteractionSet class` *InteractionSet class and constructors*

Description

The InteractionSet class stores information about pairwise genomic interactions, and is intended for use in data analysis from Hi-C or ChIA-PET experiments. Each row of the InteractionSet corresponds to a pairwise interaction between two loci, as defined in the GInteractions object. Each column corresponds to a library or sample. Each InteractionSet also contains one or more assays, intended to hold experimental data about interaction frequencies for each interaction in each sample.

Usage

```
## S4 method for signature 'ANY,GInteractions'
InteractionSet(assays, interactions, ...)

## S4 method for signature 'missing,missing'
InteractionSet(assays, interactions, ...)
```

Arguments

`assays`	A numeric matrix or a list or `SimpleList` of matrices, containing data for each interaction.
`interactions`	A GInteractions object of length equal to the number of rows in `assays`.
`...`	Other arguments to be passed to `SummarizedExperiment`.

Details

The InteractionSet class inherits from the SummarizedExperiment class and has access to all of its data members and methods (see ?`SummarizedExperiment-class` for more details). It also contains an additional `interactions` slot which holds a GInteractions object (or an object from any derived classes, e.g., StrictGInteractions). Each row of the InteractionSet object corresponds to a pairwise interaction between two genomic loci in `interactions`.

The constructor will return an InteractionSet object containing all of the specified information - for `InteractionSet,missing,missing-method`, an empty InteractionSet object is returned. Note that any `metadata` arguments will be placed in the metadata of the internal SummarizedExperiment object, *not* the metadata of the internal GInteractions object. This is consistent with the behaviour of similar classes like RangedSummarizedExperiment.

Value

For the constructors, an InteractionSet object is returned.

Author(s)

Aaron Lun

See Also

SummarizedExperiment, interaction-access, interaction-subset, interaction-compare, SummarizedExperiment-class

Examples

```
example(GInteractions, echo=FALSE)
Nlibs <- 4
counts <- matrix(rpois(Np*Nlibs, lambda=10), ncol=Nlibs)
colnames(counts) <- seq_len(Nlibs)

iset <- InteractionSet(counts, gi)
iset <- InteractionSet(counts, gi, colData=DataFrame(lib.size=1:Nlibs*1000))
iset <- InteractionSet(counts, gi, metadata=list(name="My Hi-C data"))

# Note differences in metadata storage:
metadata(iset)
metadata(interactions(iset))
```

Linearize interactions

Linearize 2D interactions into 1D ranges

Description

Convert interactions in two-dimensional space to one-dimensional ranges on the linear genome.

Usage

```
## S4 method for signature 'GInteractions,numeric'
linearize(x, ref, internal=TRUE)

## S4 method for signature 'GInteractions,GRanges'
linearize(x, ref, ..., internal=TRUE)

# Equivalent calls for InteractionSet objects.
```

Arguments

x	A GInteractions or InteractionSet object
ref	A numeric vector or a GRanges object, specifying the reference region(s) to use for linearization. If numeric, the entries should be indices pointing to a genomic interval in regions(x).
internal	A logical scalar specifying whether interactions within ref should be reported.
...	Other arguments, passed to overlapsAny in the GRanges methods.

Details

This method identifies all interactions with at least one anchor region overlapping the specified region(s) in `ref`. When `x` is a GInteractions object, the method returns a GRanges object with one entry per identified interaction, where the coordinates are defined as the *other* anchor region, i.e., the one that does *not* overlap the reference region.

If both of the anchor regions for an interaction overlap the reference regions, the genomic interval spanned by both anchor regions is returned. This is because it is not clear which region should be defined as the "other" anchor in such circumstances. Note that this will fail if the reference regions occur across multiple chromosomes. If `internal=FALSE`, interactions with both overlapping anchor regions are removed from the output.

When `x` is an InteractionSet object, a RangedSummarizedExperiment object is returned. Each entry corresponds to an identified interaction with the non-overlapping anchor region stored in the `rowRanges`. Experimental data associated with each identified interaction is stored in the various assays.

This method effectively converts two-dimesional interaction data into one-dimensional coverage across the linear genome. It is useful when a particular genomic region is of interest - this can be used as `ref`, to examine the behaviour of all other regions relative to it. For example, Hi-C data in `x` can be converted into pseudo-4C contact frequencies after linearization.

Disjoint ranges across multiple chromosomes are supported when `ref` is a GRanges object. However, it usually only makes sense to use continguous ranges as a single bait region. Similarly, if `ref` is numeric, it should refer to consecutive entries in `regions(x)` to specify the bait region.

Value

A GRanges when `x` is a GInteractions object, and a RangedSummarizedExperiment when `x` is an InteractionSet object.

Examples

```
example(InteractionSet, echo=FALSE)

# With integers
out <- linearize(iset, 1)
linearize(iset, 10)
linearize(iset, 20)

# With ranges
linearize(iset, regions(iset)[1], type="equal")
linearize(iset, regions(iset)[10], type="equal")
linearize(iset, regions(iset)[20], type="equal")
```

`linkOverlaps` *Link overlapping regions*

Description

Identify interactions that link two sets of regions by having anchor regions overlapping one entry in each set.

Usage

```
## S4 method for signature 'GInteractions,Vector,Vector'
linkOverlaps(query, subject1, subject2, ...,
    ignore.strand=TRUE, use.region="both")
```

Arguments

query	A GInteractions or InteractionSet object.
subject1, subject2	A Vector object defining a set of genomic intervals, such as a GRanges or GRangesList. subject2 may be missing. Alternatively, both subject1 and subject2 may be Hits objects, see below.
..., ignore.strand	Additional arguments to be passed to findOverlaps. Note that ignore.strand=TRUE by default, as genomic interactions are usually unstranded.
use.region	A string specifying which query regions should be used to overlap which subject. Ignored if subject2 is missing.

Details

This function identifies all interactions in query where one anchor overlaps an entry in subject1 and the other anchor overlaps an entry in subject2. It is designed to be used to identify regions that are linked by interactions in query. For example, one might specify genes as subject1 and enhancers as subject2, to identify all gene-enhancer contacts present in query. This is useful when the exact pairings between subject1 and subject2 are undefined.

The function returns a DataFrame specifying the index of the interaction in query; the index of the overlapped region in subject1; and the index of the overlapped region in subject2. If multiple regions in subject1 and/or subject2 are overlapping the anchor regions of a particular interaction, all combinations of two overlapping regions (one from each subject* set) are reported for that interaction.

By default, use.region="both" such that overlaps will be considered between any first/second interacting region in query and either subject1 or subject2. If use.region="same", overlaps will only be considered between the first interacting region in query and entries in subject1, and between the second interacting region and subject2. The opposite applies with use.region="reverse", where the first and second interacting regions are overlapped with subject2 and subject1 respectively.

If subject2 is not specified, links within subject1 are identified instead, i.e., subject2 is set to subject1. In such cases, the returned DataFrame is such that the first subject index is always greater than the second subject index, to avoid redundant permutations.

Value

A DataFrame of integer indices indicating which elements of `query` link which elements of `subject1` and `subject2`.

Using Hits as input

Hits objects can be used for the `subject1` and `subject2` arguments. These should be constructed using `findOverlaps` with `regions(query)` as the query and the genomic regions of interest as the subject. For example, the calls below:

```
> linkOverlaps(query, subject1) # 1
> linkOverlaps(query, subject1, subject2) # 2
```

will yield exactly the same output as:

```
> olap1 <- findOverlaps(regions(query), subject1)
> linkOverlaps(query, olap1) # 1
> olap2 <- findOverlaps(regions(query), subject2)
> linkOverlaps(query, olap1, olap2) # 2
```

This is useful in situations where `regions(query)` and the genomic regions in `subject1` and `subject2` are constant across multiple calls to `linkOverlaps`. In such cases, the overlaps only need to be calculated once, avoiding redundant work within `linkOverlaps`.

Author(s)

Aaron Lun

See Also

`findOverlaps,GInteractions,Vector-method`

Examples

```
example(GInteractions, echo=FALSE)

all.genes <- GRanges("chrA", IRanges(0:9*10, 1:10*10))
all.enhancers <- GRanges("chrB", IRanges(0:9*10, 1:10*10))

out <- linkOverlaps(gi, all.genes, all.enhancers)
head(out)

out <- linkOverlaps(gi, all.genes)
head(out)

# Same methods apply for InteractionSet objects.

example(InteractionSet, echo=FALSE)
out <- linkOverlaps(iset, all.genes, all.enhancers)
out <- linkOverlaps(iset, all.genes)
```

pairs *Extract paired ranges*

Description

Represent interactions in a GInteractions or Interaction object as a Pairs, SelfHits or GRangesList object.

Usage

```
## S4 method for signature 'GInteractions'
pairs(x, id=FALSE, as.grlist=FALSE)

# Equivalent call for InteractionSet to above.

makeGInteractionsFromGRangesPairs(x)
```

Arguments

x	For pairs, a GInteractions or InteractionSet object. For makeGInteractionsFromGRangesPairs, a Pairs object containing two parallel GRanges.
id	A logical scalar specifying whether indices should be returned instead of regions.
as.grlist	A logical scalar specifying whether a GRangesList should be returned.

Details

Recall that the GInteractions object stores anchor regions for each interaction in two parallel GRanges, where corresponding entries between the two GRanges constitute the pair of regions for one interaction. These parallel ranges can be extracted and stored as a Pairs object for further manipulation. This is similar to the GRangesList reported by anchors with type="both" and id=FALSE. The reverse conversion is performed using makeGInteractionsFromGRangesPairs.

An alternative representation involves storing the two anchors for each interaction in a single GRanges of length 2. Multiple interactions are then stored as a GRangesList, along with any metadata and sequence information. This is returned if as.grlist=FALSE, may be more useful in some applications where the two interacting regions must be in the same GRanges. Finally, if id=TRUE, the anchor indices are extracted and returned as a SelfHits object. This may be useful for graph construction.

Value

For pairs, if id=TRUE, a SelfHits object is returned. Otherwise, if as.grlist=TRUE, a GRangesList object is returned. Otherwise, a Pairs object is returned.

For makeGInteractionsFromGRangesPairs, a GInteractions object is returned.

Author(s)

Aaron Lun

See Also

GInteractions, Pairs, SelfHits, GRangesList

Examples

```
example(GInteractions, echo=FALSE)
y <- pairs(gi)
y
makeGInteractionsFromGRangesPairs(y)

pairs(gi, id=TRUE)
pairs(gi, as.grlist=TRUE)

example(InteractionSet, echo=FALSE)
pairs(iset)
pairs(iset, id=TRUE)
pairs(iset, as.grlist=TRUE)
```

---

updateObject *Update a GInteractions object*

Usage

```
## S4 method for signature 'GInteractions'
updateObject(object, ..., verbose = FALSE)
```

Arguments

object	A old GInteractions object.
...	Additional arguments that are ignored.
verbose	Logical scalar indicating whether a message should be emitted as the object is updated.

Value

An updated version of object.

Index