

Package ‘RaggedExperiment’

October 18, 2017

Title Representation of Sparse Experiments and Assays Across Samples

Version 1.0.0

Description This package provides a flexible representation of copy number, mutation, and other data that fit into the ragged array schema for genomic location data. The basic representation of such data provides a rectangular flat table interface to the user with range information in the rows and samples/specimen in the columns.

License Artistic-2.0

biocViews Infrastructure, DataRepresentation

BugReports <https://github.com/Bioconductor/RaggedExperiment/issues>

VignetteBuilder knitr

Depends R (>= 3.4.0), GenomicRanges

Imports BiocGenerics, IRanges, methods, S4Vectors, stats, SummarizedExperiment

Suggests knitr, testthat, MultiAssayExperiment

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Martin Morgan [aut, cre],
Marcel Ramos [aut]

Maintainer Martin Morgan <martin.morgan@roswellpark.org>

R topics documented:

RaggedExperiment-package	2
assay-functions	2
RaggedExperiment-class	5
sparseSummarizedExperiment	8

Index	11
--------------	-----------

RaggedExperiment-package

RaggedExperiment: Range-based data representation package

Description

[RaggedExperiment](#) allows the user to represent, copy number, mutation, and other types of range-based data formats where optional information about samples can be provided. At the backbone of this package is the [GRangesList](#) class. The [RaggedExperiment](#) class uses this representation and presents the data in a couple of different ways:

- [rowRanges](#)
- [colData](#)

The [rowRanges](#) method will return the internal [GRangesList](#) representation of the dataset. A distinction between the [GRangesList](#) and the [RaggedExperiment](#) classes is that the [RaggedExperiment](#) class allows for ragged ranges, meaning that there may be a different number of ranges for each sample.

Author(s)

Maintainer: Martin Morgan <martin.morgan@roswellpark.org>

Authors:

- Marcel Ramos <marcel.ramos@roswellpark.org>

See Also

Useful links:

- Report bugs at <https://github.com/Bioconductor/RaggedExperiment/issues>

assay-functions

Create simplified representation of ragged assay data.

Description

These methods transform [assay\(\)](#) from the default (i.e., [sparseAssay\(\)](#)) representation to various forms of more dense representation. [compactAssay\(\)](#) collapses identical ranges across samples into a single row. [disjoinAssay\(\)](#) creates disjoint (non-overlapping) regions, simplifies values within each sample in a user-specified manner, and returns a matrix of disjoint regions x samples.

This method transforms [assay\(\)](#) from the default (i.e., [sparseAssay\(\)](#)) representation to a reduced representation summarizing each original range overlapping ranges in query. Reduction in each cell can be tailored to individual needs using the [simplify](#) argument.

Usage

```

sparseAssay(x, i = 1, withDimnames = TRUE, background = NA)

compactAssay(x, i = 1, withDimnames = TRUE, background = NA)

disjoinAssay(x, simplify, i = 1, withDimnames = TRUE, background = NA)

qreduceAssay(x, query, simplify, i = 1, withDimnames = TRUE,
             background = NA)

```

Arguments

<code>x</code>	A <code>RaggedExperiment</code> object
<code>i</code>	<code>integer(1)</code> or <code>character(1)</code> name of assay to be transformed.
<code>withDimnames</code>	<code>logical(1)</code> include dimnames on the returned matrix. When there are no explicit rownames, these are manufactured with <code>as.character(rowRanges(x))</code> ; rownames are always manufactured for <code>compactAssay()</code> and <code>disjoinAssay()</code> .
<code>background</code>	A value (default NA) for the returned matrix after <code>*Assay</code> operations
<code>simplify</code>	<code>disjoinAssay</code> : A function operating on a <code>*List</code> , where the elements of the list are all within-sample assay values from ranges overlapping each disjoint range. For instance, to use the <code>simplify=mean</code> of overlapping ranges, where ranges are characterized by integer-valued scores, the entries are calculated as

```

          a
original: |-----|
          b
          |-----|

          a  a, b  b
disjoint: |----|-----|----|

values <- IntegerList(a, c(a, b), b)
simplify(values)

```

`qreduceAssay`: A function accepting arguments `score`, `range`, and `qrangle`:

- `score` A `*List`, where each list element corresponds to a cell in the matrix to be returned by `qreduceAssay`. Vector elements correspond to ranges overlapping query. The `*List` objects support many vectorized mathematical operations, so `simplify` can be implemented efficiently.
- `range` A `GRangesList` instance, 'parallel' to `score`. Each element of the list corresponds to a cell in the matrix to be returned by `qreduceAssay`. Each range in the element corresponds to the range for which the score element applies.
- `qrangle` A `GRanges` instance with the same length as `score`, providing the query range to which the corresponding scores apply.

<code>query</code>	<code>GRanges</code> providing regions over which reduction is to occur.
--------------------	--

Value

`sparseAssay()`: A `matrix()` with dimensions `dim(x)`. Elements contain the assay value for the *i*th range and *j*th sample.

`compactAssay()`: Samples with identical range are placed in the same row. Non-disjoint ranges are NOT collapsed.

`disjoinAssay()`: A matrix with number of rows equal to number of disjoint ranges across all samples. Elements of the matrix are summarized by applying `simplify()` to assay values of overlapping ranges

`qreduceAssay()`: A matrix() with dimensions `length(query) x ncol(x)`. Elements contain assay values for the *i*th query range and *j*th sample, summarized according to the function `simplify`.

Examples

```
x <- RaggedExperiment(GRangesList(
  GRanges(c("A:1-3", "A:4-5", "A:10-15"), score=1:3),
  GRanges(c("A:4-5", "B:1-3"), score=4:5)
))
query <- GRanges(c("A:1-2", "A:4-5", "B:1-5"))

weightedmean <- function(scores, ranges, qranges)
  ## weighted average score per query range
  sum(scores * width(ranges)) / sum(width(ranges))
qreduceAssay(x, query, weightedmean)

## Not run:
## Extended example: non-silent mutations, summarized by genic
## region

suppressPackageStartupMessages({
  library(TxDb.Hsapiens.UCSC.hg19.knownGene)
  library(org.Hs.eg.db)
  library(GenomeInfoDb)
  library(MultiAssayExperiment)
})

## TCGA Multi-assay experiment to RaggedExperiment

url <- "http://s3.amazonaws.com/multiassayexperiments/accMAE0.rds"
## download.file(url, fl <- tempfile())
## fl <- "accMAE0.rds"
mae <- readRDS(fl)[, , c("RNASeq2GeneNorm", "CNASNP", "Mutations")]

## genomic coordinates

gn <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)
gn <- keepStandardChromosomes(granges(gn), pruning.mode="coarse")
seqlevelsStyle(gn) <- "NCBI"

## reduce mutations, marking any genomic range with non-silent
## mutation as FALSE

nonsilent <- function(scores, ranges, qranges)
  any(scores != "Silent")
re <- as(mae[["Mutations"]], "RaggedExperiment")
mutations <- qreduceAssay(re, gn, nonsilent, "Variant_Classification")

## reduce copy number

re <- as(mae[["CNASNP"]], "RaggedExperiment")
```

```
cn <- qreduceAssay(re, gn, weightedmean, "Segment_Mean")  
## End(Not run)
```

RaggedExperiment-class

RaggedExperiment objects

Description

The `RaggedExperiment` class is a container for storing range-based data, including but not limited to copy number data, and mutation data. It can store a collection of `GRanges` objects, as it is derived from the `GenomicRangesList`.

Usage

```
RaggedExperiment(..., colData = DataFrame())  
  
## S4 method for signature 'RaggedExperiment'  
rowRanges(x, ...)  
  
## S4 method for signature 'RaggedExperiment'  
dim(x)  
  
## S4 method for signature 'RaggedExperiment'  
dimnames(x)  
  
## S4 replacement method for signature 'RaggedExperiment,list'  
dimnames(x) <- value  
  
## S4 method for signature 'RaggedExperiment'  
colData(x, ...)  
  
## S4 method for signature 'RaggedExperiment,missing'  
assay(x, i, ...)  
  
## S4 method for signature 'RaggedExperiment,ANY'  
assay(x, i, ..., withDimnames = TRUE)  
  
## S4 method for signature 'RaggedExperiment'  
assays(x, ..., withDimnames = TRUE)  
  
## S4 method for signature 'RaggedExperiment'  
assayNames(x, ...)  
  
## S4 method for signature 'RaggedExperiment'  
show(object)  
  
## S4 method for signature 'RaggedExperiment,ANY,ANY,ANY'  
x[i, j, ..., drop = TRUE]
```

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

## S4 method for signature 'RaggedExperiment,Vector'
subsetByOverlaps(query, subject,
  maxgap = 0L, minoverlap = 1L, type = c("any", "start", "end", "within",
  "equal"), invert = FALSE, ...)
```

Arguments

...	Constructor: GRanges, list of GRanges, or GRangesList OR assay: Additional arguments for assay. See details for more information.
colData	A DataFrame describing samples. Length of rowRanges must equal the number of rows in colData
x	A RaggedExperiment object.
value	A list of dimension names
i	logical(1), integer(1), or character(1) indicating the assay to be reported. For [, i can be any supported Vector object, e.g., GRanges.
withDimnames	logical (default TRUE) whether to use dimension names in the resulting object
object	A RaggedExperiment object.
j	integer(), character(), or logical() index selecting columns from RaggedExperiment
drop	logical (default TRUE) whether to drop empty samples
query	A RaggedExperiment instance.
subject	Each of them can be a Ranges , Views , RangesList , ViewsList , or RangedData object. In addition, if subject is a Ranges object, query can be an integer vector to be converted to length-one ranges. If query is a RangesList or RangedData , subject must be a RangesList or RangedData . If both lists have names, each element from the subject is paired with the element from the query with the matching name, if any. Otherwise, elements are paired by position. The overlap is then computed between the pairs as described below. If subject is omitted, query is queried against itself. In this case, and only this case, the drop.self and drop.redundant arguments are allowed. By default, the result will contain hits for each range against itself, and if there is a hit from A to B, there is also a hit for B to A. If drop.self is TRUE, all self matches are dropped. If drop.redundant is TRUE, only one of A->B and B->A is returned.
maxgap	Intervals with a separation of maxgap or less and a minimum of minoverlap overlapping positions, allowing for maxgap, are considered to be overlapping. maxgap should be a scalar, non-negative, integer. minoverlap should be a scalar, positive integer.
minoverlap	Intervals with a separation of maxgap or less and a minimum of minoverlap overlapping positions, allowing for maxgap, are considered to be overlapping. maxgap should be a scalar, non-negative, integer. minoverlap should be a scalar, positive integer.
type	By default, any overlap is accepted. By specifying the type parameter, one can select for specific types of overlap. The types correspond to operations in

Allen's Interval Algebra (see references). If `type` is `start` or `end`, the intervals are required to have matching starts or ends, respectively. While this operation seems trivial, the naive implementation using `outer` would be much less efficient. Specifying `equal` as the type returns the intersection of the start and end matches. If `type` is `within`, the query interval must be wholly contained within the subject interval. Note that all matches must additionally satisfy the `minoverlap` constraint described above.

The `maxgap` parameter has special meaning with the special overlap types. For `start`, `end`, and `equal`, it specifies the maximum difference in the starts, ends or both, respectively. For `within`, it is the maximum amount by which the subject may be wider than the query.

`invert` If `TRUE`, keep only the query ranges that do *not* overlap the subject.

Value

constructor returns a `RaggedExperiment` object

'`rowRanges`' returns a `GRanges` object summarizing ranges corresponding to `assay()` rows.

'`assays`' returns a `SimpleList`

`overlapsAny` returns a logical vector of length equal to the number of rows in the query; `TRUE` when the copy number region overlaps the subject.

`subsetByOverlaps` returns a `RaggedExperiment` containing only copy number regions overlapping subject.

Methods (by generic)

- `rowRanges`: `rowRanges` accessor
- `dim`: get dimensions (number of sample-specific row ranges by number of samples)
- `dimnames`: get row (sample-specific) range names and sample names
- `dimnames<-`: set row (sample-specific) range names and sample names
- `colData`: get column data
- `assay`: assay missing method uses first metadata column
- `assay`: assay numeric method.
- `assays`: assays
- `assayNames`: names in each assay
- `show`: show method
- `[]`: Subset a `RaggedExperiment` object
- `overlapsAny`: Determine whether copy number ranges defined by query overlap ranges of subject.
- `subsetByOverlaps`: Subset the `RaggedExperiment` to contain only copy number ranges overlapping ranges of subject.

Constructors

`RaggedExperiment(..., colData=DataFrame())`: Creates a `RaggedExperiment` object using multiple `GRanges` objects or a list of `GRanges` objects. Additional column data may be provided as a `DataFrame` object.

Subsetting

In the following, 'x' represents a `RaggedExperiment` object:

`x[i, j]`: Get ranges or elements (i and j, respectively) with optional metadata columns where i or j can be missing, an NA-free logical, numeric, or character vector.

Coercion

Coercion possible from `RangedRaggedAssay` to `RaggedExperiment`. Here object represents a `RangedRaggedAssay`: `as(object, "RaggedExperiment")`

Examples

```
## Create an empty RaggedExperiment instance
re0 <- RaggedExperiment()
re0

## Create a couple of GRanges objects with row ranges names
sample1 <- GRanges(
  c(a = "chr1:1-10:-", b = "chr1:11-18:"),
  score = 1:2)
sample2 <- GRanges(
  c(c = "chr2:1-10:-", d = "chr2:11-18:"),
  score = 3:4)

## Include column data
colDat <- DataFrame(id = 1:2)

## Create a RaggedExperiment object from a couple of GRanges
re1 <- RaggedExperiment(sample1=sample1, sample2=sample2, colData = colDat)
re1

## With list of GRanges
lgr <- list(sample1 = sample1, sample2 = sample2)

## Create a RaggedExperiment from a list of GRanges
re2 <- RaggedExperiment(lgr, colData = colDat)

gr1 <- GRangesList(sample1 = sample1, sample2 = sample2)

## Create a RaggedExperiment from a GRangesList
re3 <- RaggedExperiment(gr1, colData = colDat)

## Subset a RaggedExperiment
assay(re3[c(1, 3),])
subsetByOverlaps(re3, GRanges("chr1:1-5")) # by ranges
```

sparseSummarizedExperiment

Create SummarizedExperiment representations by transforming ragged assays to rectangular form.

Description

These methods transform `RaggedExperiment` objects to similar `SummarizedExperiment` objects. They do so by transforming assay data to more rectangular representations, following the rules outlined for similarly named transformations `sparseAssay()`, `compactAssay()`, `disjoinAssay()`, and `qreduceAssay()`. Because of the complexity of the transformation, it usually only makes sense to transform `RaggedExperiment` objects with a single assay; this is currently enforced at time of coercion.

Usage

```
sparseSummarizedExperiment(x, i = 1, withDimnames = TRUE)

compactSummarizedExperiment(x, i = 1L, withDimnames = TRUE)

disjoinSummarizedExperiment(x, simplify, i = 1L, withDimnames = TRUE)

qreduceSummarizedExperiment(x, query, simplify, i = 1L, withDimnames = TRUE)
```

Arguments

<code>x</code>	<code>RaggedExperiment</code>
<code>i</code>	<code>integer(1)</code> , <code>character(1)</code> , or <code>logical()</code> selecting the assay to be transformed.
<code>withDimnames</code>	<code>logical(1)</code> default <code>TRUE</code> . propagate dimnames to <code>SummarizedExperiment</code> .
<code>simplify</code>	function of 1 (for <code>disjoinSummarizedExperiment</code>) or 3 (for <code>qreduceSummarizedExperiment</code>) arguments, used to transform assays. See assay-functions .
<code>query</code>	<code>GRanges</code> providing regions over which reduction is to occur.

Value

All functions return `RangedSummarizedExperiment`.

`sparseSummarizedExperiment` has `rowRanges()` identical to the row ranges of `x`, and `assay()` data as `sparseAssay()`. This is a very space-inefficient representation of ragged data.

`compactSummarizedExperiment` has `rowRanges()` identical to the row ranges of `x`, and `assay()` data as `compactAssay()`. This is a space-inefficient representation of ragged data when samples are primarily composed of different ranges.

`disjoinSummarizedExperiment` has `rowRanges()` identical to the disjoint row ranges of `x`, `disjoint(rowRanges(x))`, and `assay()` data as `disjoinAssay()`.

`qreduceSummarizedExperiment` has `rowRanges()` identical to `query`, and `assay()` data as `qreduceAssay()`.

Examples

```
x <- RaggedExperiment(GRangesList(
  GRanges(c("A:1-5", "A:4-6", "A:10-15"), score=1:3),
  GRanges(c("A:1-5", "B:1-3"), score=4:5)
))

## sparseSummarizedExperiment

sse <- sparseSummarizedExperiment(x)
assay(sse)
```

```
rowRanges(sse)

## compactSummarizedExperiment

cse <- compactSummarizedExperiment(x)
assay(cse)
rowRanges(cse)

## disjointSummarizedExperiment

disjoinAssay(x, lengths)
dse <- disjointSummarizedExperiment(x, lengths)
assay(dse)
rowRanges(dse)

## qreduceSummarizedExperiment

x <- RaggedExperiment(GRangesList(
  GRanges(c("A:1-3", "A:4-5", "A:10-15"), score=1:3),
  GRanges(c("A:4-5", "B:1-3"), score=4:5)
))
query <- GRanges(c("A:1-2", "A:4-5", "B:1-5"))

weightedmean <- function(scores, ranges, qranges)
  ## weighted average score per query range
  sum(scores * width(ranges)) / sum(width(ranges))

qreduceAssay(x, query, weightedmean)
qse <- qreduceSummarizedExperiment(x, query, weightedmean)
assay(qse)
rowRanges(qse)
```

Index

- [,RaggedExperiment,ANY,ANY,ANY-method
(RaggedExperiment-class), 5
- assay,RaggedExperiment,ANY-method
(RaggedExperiment-class), 5
- assay,RaggedExperiment,missing-method
(RaggedExperiment-class), 5
- assay-functions, 2
- assayNames,RaggedExperiment-method
(RaggedExperiment-class), 5
- assays,RaggedExperiment-method
(RaggedExperiment-class), 5
- class:RaggedExperiment
(RaggedExperiment-class), 5
- colData,RaggedExperiment-method
(RaggedExperiment-class), 5
- compactAssay (assay-functions), 2
- compactSummarizedExperiment
(sparseSummarizedExperiment), 8
- DataFrame, 6
- dim,RaggedExperiment-method
(RaggedExperiment-class), 5
- dimnames,RaggedExperiment-method
(RaggedExperiment-class), 5
- dimnames<-,RaggedExperiment,list-method
(RaggedExperiment-class), 5
- disjoinAssay (assay-functions), 2
- disjoinSummarizedExperiment
(sparseSummarizedExperiment), 8
- GRanges, 7
- GRangesList, 2
- overlapsAny,RaggedExperiment,Vector-method
(RaggedExperiment-class), 5
- qreduceAssay (assay-functions), 2
- qreduceSummarizedExperiment
(sparseSummarizedExperiment), 8
- RaggedExperiment, 2
- RaggedExperiment
(RaggedExperiment-class), 5
- RaggedExperiment-class, 5
- RaggedExperiment-package, 2
- RangedData, 6
- RangedRaggedAssay, 8
- Ranges, 6
- RangesList, 6
- rowRanges, 2
- rowRanges,RaggedExperiment-method
(RaggedExperiment-class), 5
- show,RaggedExperiment-method
(RaggedExperiment-class), 5
- SimpleList, 7
- sparseAssay (assay-functions), 2
- sparseSummarizedExperiment, 8
- subsetByOverlaps,RaggedExperiment,Vector-method
(RaggedExperiment-class), 5
- Views, 6
- ViewsList, 6