Package 'SplicingGraphs'

November 12, 2026

Title Create, manipulate, visualize splicing graphs, and assign RNA-seq reads to them

Description This package allows the user to create, manipulate, and visualize splicing graphs and their bubbles based on a gene model for a given organism. Additionally it allows the user to assign RNA-seq reads to the edges of a set of splicing graphs, and to summarize them in different ways.

biocViews Genetics, Annotation, DataRepresentation, Visualization, Sequencing, RNASeq, GeneExpression, AlternativeSplicing, Transcription, ImmunoOncology

URL https://bioconductor.org/packages/SplicingGraphs

BugReports https://github.com/Bioconductor/SplicingGraphs/issues

Version 1.52.0

License Artistic-2.0

Encoding UTF-8

Author D. Bindreither, M. Carlson, M. Morgan, H. Pagès

Maintainer H. Pagès <hpages.on.github@gmail.com>

Depends GenomicFeatures (>= 1.17.13), GenomicAlignments (>= 1.1.22), Rgraphviz (>= 2.3.7)

Imports methods, utils, graphics, igraph, BiocGenerics, S4Vectors (>= 0.17.5), BiocParallel, IRanges (>= 2.21.2), Seqinfo, GenomicRanges (>= 1.23.21), Rsamtools, graph

Suggests igraph, Gviz, txdbmaker, TxDb.Hsapiens.UCSC.hg19.refGene, RNAseqData.HNRNPC.bam.chr14, RUnit

Collate utils.R igraph-utils.R SplicingGraphs-class.R plotTranscripts-methods.R sgedgesByGene-methods.R txpath-methods.R sgedges-methods.R sgraph-methods.R rsgedgesByGene-methods.R bubbles-methods.R assignReads.R countReads-methods.R toy_data.R zzz.R

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

git_branch RELEASE_3_23

git_last_commit d30b5cf

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

SplicingGraphs-package 2
assignReads 4
bubbles-methods 7
countReads-methods 8
plotTranscripts-methods 11
rsgedgesByGene-methods 13
sgedges-methods 15
sgedgesByGene-methods 17
sgraph-methods 20
SplicingGraphs-class 21
toy_data 25
TSPCsg 27
txpath-methods 28

Index **32**

`SplicingGraphs-package`

Create, manipulate, visualize splicing graphs, and assign RNA-seq reads to them

---

Description

The **SplicingGraphs** package allows the user to create, manipulate, and visualize splicing graphs and their bubbles based on a gene model for a given organism. Additionally it allows the user to assign RNA-seq reads to the edges of a set of splicing graphs, and to summarize them in different ways.

Details

See the *Splicing graphs and RNA-seq data* vignette in the package for a gentle introduction to its use. To access the vignette, do `browseVignettes("SplicingGraphs")`, then click on the link to the PDF version.

Note

The **SplicingGraphs** package is a reincarnation of an internal project, the **SpliceGraph** package, originally written by D. Bindreither, M. Carlson, and M. Morgan. **SpliceGraph** was never released as part of Bioconductor.

With respect to the old **SpliceGraph**, the scope of the new **SplicingGraphs** package has been reduced to focus only on the following functionalities: creating/manipulating/plotting splicing graphs, computing the bubbles and AS codes, and assigning/counting reads.

In addition to this, the old **SpliceGraph** package also had facilities for performing some downstream statistical analysis. They were covered in its vignette under the following topics/sections:

- Experimental design
- Significant altered alternative splice events
- Modification of GLM employed in the **DEXSeq** package
- Differential edge expression analysis
- Comparison to the classic **DEXSeq** analysis

The **SplicingGraphs** vignette doesn't cover any of this and the new package provides no facilities for doing this type of downstream statistical analysis.

Author(s)

Author and maintainer: H. Pagès <hpages.on.github@gmail.com>

The **SplicingGraphs** package is a complete revamp (design and implementation) of the old **SpliceGraph** package (see Note above).

References

Heber, S., Alekseyev, M., Sze, S., Tang, H., and Pevzner, P. A. *Splicing graphs and EST assembly problem* Bioinformatics Date: Jul 2002 Vol: 18 Pages: S181-S188

Sammeth, M. (2009) *Complete Alternative Splicing Events Are Bubbles in Splicing Graphs* J. Comput. Biol. Date: Aug 2009 Vol: 16 Pages: 1117-1140

See Also

The man pages in the **SplicingGraphs** package are:

1. The SplicingGraphs class.
2. `plotTranscripts` for plotting a set of transcripts along genomic coordinates.
3. `sgedgesByGene` for extracting the edges and their ranges from a SplicingGraphs object.
4. `txpath` for extracting the transcript paths of a splicing graph.
5. `sgedges` for extracting the edges (and nodes) of a splicing graph.
6. `sgraph` for extracting a splicing graph as a plottable graph-like object.
7. `rsgedgesByGene` for extracting the reduced edges and their ranges from a SplicingGraphs object.
8. `bubbles` for computing the bubbles of a splicing graph.

9. assignReads for assigning reads to the edges of a SplicingGraphs object.
10. countReads for summarizing the reads assigned to a SplicingGraphs object.
11. toy_genes_gff for details about the toy data included in this package.

Examples

```
if (interactive()) {
  ## Access the "Splicing graphs and RNA-seq data" vignette with:
  browseVignettes("SplicingGraphs")
}
```

---

assignReads *Assign reads to the edges of a SplicingGraphs object*

Description

assignReads assigns reads to the exonic and intronic edges of a SplicingGraphs object.

removeReads removes all the reads assigned to the exonic and intronic edges of a SplicingGraphs object.

Usage

```
assignReads(sg, reads, sample.name=NA)

removeReads(sg)
```

Arguments

sg	A SplicingGraphs object.
reads	A GAlignments, GAlignmentPairs, or GRangesList object, containing the reads to assign to the exons and introns in sg. It must have unique names on it, typically the QNAME ("query name") field coming from the BAM file. More on this in the 'About the read names' section below.
sample.name	A single string containing the name of the sample where the reads are coming from.

Details

TODO

Value

For assignReads: the supplied SplicingGraphs object with the reads assigned to it.

For removeReads: the supplied SplicingGraphs object with all reads previously assigned with assignReads removed from it.

About read names

The read names are typically imported from the BAM file by calling `readGAlignments` (or `readGAlignmentPairs`) with `use.names=TRUE`. This extracts the "query names" from the file (stored in the QNAME field), and makes them the names of the returned object.

The `reads` object must have unique names on it. The presence of duplicated names generally indicates one (or both) of the following situations:

- (a) `reads` contains paired-end reads that have not been paired;
- (b) some of the reads are *secondary alignments*.

If (a): you can find out whether reads in a BAM file are single- or paired-end with the `quickBamFlagSummary` utility from the **Rsamtools** package. If they're paired-end, load them with `readGAlignmentPairs` instead of `readGAlignments`, and that will pair them.

If (b): you can filter out secondary alignments by passing `'isSecondaryAlignment=FALSE'` to `scanBamFlag` when preparing the ScanBamParam object used to load the reads. For example:

```
flag0 <- scanBamFlag(isSecondaryAlignment=FALSE,
                     isNotPassingQualityControls=FALSE,
                     isDuplicate=FALSE)
param0 <- ScanBamParam(flag=flag0)
reads <- readGAlignments("path/to/BAM/file", use.names=TRUE,
                                param=param0)
```

This will filter out records that have flag 0x100 (secondary alignment) set to 1. See `?scanBamFlag` in the **Rsamtools** package for more information. See the SAM Specs on the SAMtools project page at `http://samtools.sourceforge.net/` for a description of the SAM/BAM flags.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see `?` `SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Other topics related to this man page and documented in other packages:

- The GRangesList class defined in the **GenomicRanges** package.
- The GAlignments and GAlignmentPairs classes, as well as the `readGAlignments` and `readGAlignmentPairs` functions defined in the **GenomicAlignments** package.
- The `quickBamFlagSummary` and `ScanBamParam` functions defined in the **Rsamtools** package.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model (see
##    '?SplicingGraphs')
## ---------------------------------------------------------------------
example(SplicingGraphs)
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## ---------------------------------------------------------------------
## 2. Load toy reads
## ---------------------------------------------------------------------
## Load toy reads (single-end) from a BAM file. We filter out secondary
## alignments, reads not passing quality controls, and PCR or optical
## duplicates (see ?scanBamFlag in the Rsamtools package for more
## information):
flag0 <- scanBamFlag(isSecondaryAlignment=FALSE,
                     isNotPassingQualityControls=FALSE,
                     isDuplicate=FALSE)
param0 <- ScanBamParam(flag=flag0)
gal <- readGAlignments(toy_reads_bam(), use.names=TRUE, param=param0)
gal

## ---------------------------------------------------------------------
## 3. Assign the reads to the exons and introns in 'sg'
## ---------------------------------------------------------------------
## The same read can be assigned to more than 1 exon or intron (e.g. a
## junction read with 1 gap can be assigned to 2 exons and 1 intron).
sg <- assignReads(sg, gal, sample.name="TOYREADS")

## See the assignments to the splicing graph edges.
edge_by_tx <- sgedgesByTranscript(sg, with.hits.mcols=TRUE)
edge_data <- mcols(unlist(edge_by_tx))
colnames(edge_data)
head(edge_data)
edge_data[ , c("sgedge_id", "TOYREADS.hits")]

edge_by_gene <- sgedgesByGene(sg, with.hits.mcols=TRUE)
mcols(unlist(edge_by_gene))

## See the assignments to the reduced splicing graph edges.
redge_by_gene <- rsgedgesByGene(sg, with.hits.mcols=TRUE)
mcols(unlist(redge_by_gene))

## ---------------------------------------------------------------------
## 4. Summarize the reads assigned to 'sg' and eventually remove them
## ---------------------------------------------------------------------
## See '?countReads'.
```

bubbles-methods *Compute the bubbles of a splicing graph*

Description

bubbles computes the bubbles of the splicing graph of a given gene from a SplicingGraphs object.

Usage

```
bubbles(x)

ASCODE2DESC
```

Arguments

x	A SplicingGraphs object of length 1.

Details

TODO

Value

TODO

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
example(SplicingGraphs)  # create SplicingGraphs object 'sg'
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

plot(sgraph(sg["geneA"], tx_id.as.edge.label=TRUE))
bubbles(sg["geneA"])

plot(sgraph(sg["geneB"], tx_id.as.edge.label=TRUE))
bubbles(sg["geneB"])

plot(sgraph(sg["geneD"], tx_id.as.edge.label=TRUE))
bubbles(sg["geneD"])
```

countReads-methods *Summarize the reads assigned to a SplicingGraphs object*

Description

countReads counts the reads assigned to a SplicingGraphs object. The counting can be done by splicing graph edge, *reduced* splicing graph edge, transcript, or gene.

reportReads is similar to countReads but returns right before the final counting step, that is, the returned DataFrame contains the reads instead of their counts.

Usage

```
countReads(x, by=c("sgedge", "rsgedge", "tx", "gene"))
reportReads(x, by=c("sgedge", "rsgedge", "tx", "gene"))
```

Arguments

x	A SplicingGraphs object.
by	Can be "sgedge", "rsgedge", "tx", or "gene". Specifies the *level of resolution* that summarization should be performed at. See Details section below.

Details

Levels of resolution: countReads and reportReads allow summarization of the reads at different levels of resolution. The level of resolution is determined by the type of feature that one chooses via the by argument. The supported resolutions are (from highest to lowest resolution):

1. by="sgedge" for summarization at the splicing graph edge level (i.e. at the exons/intron level);
2. by="rsgedge" for summarization at the *reduced* splicing graph edge level;
3. by="tx" for summarization at the transcript level;
4. by="gene" for summarization at the gene level.

Relationship between levels of resolution: There is a parent-child relationship between the features corresponding to a given level of resolution (the parent features) and those corresponding to a higher level of resolution (the child features).

For example, in the case of the 2 first levels of resolution listed above, the parent-child relationship is the following: the parent features are the *reduced* splicing graph edges, the child features are the splicing graph edges, and each parent feature is obtained by merging one or more child features together. Similarly, transcripts can be seen as parent features of *reduced* splicing graph edges, and genes as parent features of transcripts. Note that, the rsgedge/sgedge and gene/tx relationships are one-to-many, but the tx/rsgedge relationship is many-to-many because a given edge can belong to more than one transcript.

Finally the parent-child relationships between 2 arbitrary levels of resolution is defined by combining the relationships between consecutive levels. All possible parent-child relationships are summarized in the following table:

```
               | to: sgedge   | to: rsgedge  | to: tx
---------------+--------------+--------------+------------
from: rsgedge  | one-to-many  |              |
from: tx       | many-to-many | many-to-many |
from: gene     | one-to-many  | one-to-many  | one-to-many
```

Multiple hits and ambiguous reads: An important distinction needs to be made between a read that hits a given feature multiple times and a read that hits more than one feature.

If the former, the read is counted/reported only once for that feature. For example, when summarizing at the transcript level, a read is counted/reported only once for a given transcript, even if that read hits more than one splicing graph edge (or *reduced* splicing graph edge) associated with that transcript.

If the latter, the read is said to be *ambiguous*. An ambiguous read is currently counted/reported for each feature where it has a hit. This is a temporary situation: in the near future the user will be offered options to handle ambiguous reads in different ways.

Ambiguous reads and levels of resolution: A read might be ambiguous at one level of resolution but not at the other. Also the number of ambiguous reads is typically affected by the level of resolution. However, even though higher resolution generally means more ambiguous reads, this is only true when the switch from one level of resolution to the other implies a parent-child relationship between features that is one-to-many. So, based on the above table, this is always true, except when switching from using `by="tx"` to using `by="sgedge"` or `by="rsgedge"`. In those cases, the switch can produce more ambiguities but it can also produce less.

Value

A DataFrame object with one row per:

- unique splicing graph edge, if `by="sgedge"`;
- unique *reduced* splicing graph edge, if `by="rsgedge"`;
- transcript if `by="tx"`;
- gene if `by="gene"`.

And with one column per sample (containing the counts for that sample for `countReads`, and the reads for that sample for `reportReads`), plus the two following left columns:

- if `by="sgedge"`: `"sgedge_id"`, containing the *global splicing graph edge ids*, and `"ex_or_in"`, containing the type of edge (exon or intron);
- if `by="rsgedge"`: `"rsgedge_id"`, containing the *global reduced splicing graph edge ids*, and `"ex_or_in"`, containing the type of edge (exon, intron, or mixed);
- if `by="tx"`: `"tx_id"` and `"gene_id"`;
- if `by="gene"`: `"gene_id"` and `"tx_id"`.

For `countReads`, each column of counts is of type integer and is named after the corresponding sample. For `reportReads`, each column of reads is a CharacterList object and its name is the name of the corresponding sample with the `".hits"` suffix added to it. In both cases, the name of the sample is the name that was passed to `assignReads` when the reads of a given sample were initially assigned. See `?assignReads` for more information.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model and assign toy
##    reads to it (see '?assignReads')
## ---------------------------------------------------------------------
example(assignReads)

## ---------------------------------------------------------------------
## 2. Summarize the reads by splicing graph edge
## ---------------------------------------------------------------------
countReads(sg)
reportReads(sg)

## ---------------------------------------------------------------------
## 3. Summarize the reads by reduced splicing graph edge
## ---------------------------------------------------------------------
countReads(sg, by="rsgedge")
reportReads(sg, by="rsgedge")

## ---------------------------------------------------------------------
## 4. Summarize the reads by transcript
## ---------------------------------------------------------------------
countReads(sg, by="tx")
reportReads(sg, by="tx")

## ---------------------------------------------------------------------
## 5. Summarize the reads by gene
## ---------------------------------------------------------------------
countReads(sg, by="gene")
reportReads(sg, by="gene")

## ---------------------------------------------------------------------
## 6. A close look at ambiguous reads
## ---------------------------------------------------------------------
resolutions <- c("sgedge", "rsgedge", "tx", "gene")

reported_reads <- lapply(resolutions,
    function(by) {
        reported_reads <- reportReads(sg, by=by)
        unlist(reported_reads$TOYREADS.hits)
    })

## The set of reported reads is the same at all levels of resolution:
```

```
unique_reported_reads <- lapply(reported_reads, unique)
stopifnot(identical(unique_reported_reads,
                    rep(unique_reported_reads[1], 4)))

## Extract ambigous reads for each level of resolution:
ambiguous_reads <- lapply(reported_reads,
                          function(x) unique(x[duplicated(x)]))
names(ambiguous_reads) <- resolutions
ambiguous_reads

## Reads that are ambiguous at the "rsgedge" level must also be
## ambiguous at the "sgedge" level:
stopifnot(all(ambiguous_reads$rsgedge %in% ambiguous_reads$sgedge))

## However, there is no reason why reads that are ambiguous at the
## "tx" level should also be ambiguous at the "sgedge" or "rsgedge"
## level!

## ---------------------------------------------------------------------
## 7. Remove the reads from 'sg'.
## ---------------------------------------------------------------------
sg <- removeReads(sg)
countReads(sg)
```

`plotTranscripts-methods` *Plot a set of transcripts along genomic coordinates.*

Description

`plotTranscripts` uses the **Gviz** package to plot the exon structure of a set of transcripts along genomic coordinates.

Usage

```
plotTranscripts(x, reads=NULL, from=NA, to=NA, max.plot.reads=200)
```

Arguments

`x`	A GRangesList object containing the genomic ranges of a set of exons grouped by transcript. Alternatively, `x` can be a TxDb object, or a SplicingGraphs object of length 1.
`reads`	A GAlignments or GAlignmentPairs object containing single-end or paired-end reads.
`from, to`	Single numeric values, giving the range of genomic coordinates to plot the tracks in. By default (i.e. `from=NA` and `to=NA`), the plot covers the range spanned by the transcripts. If `from=NULL` and `to=NULL`, then the plot covers the range spanned by the transcripts and the reads.

`max.plot.reads` The maximum number of reads that will be plotted. When the number of reads that fall in the region being plotted is very large, plotting them all would take a long time and result in a plot that is not very useful. If that number is greater than `max.plot.reads`, then only `max.plot.reads` randomly chosen reads are plotted.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Other topics related to this man page and documented in other packages:

- `plotTranscripts` is based on the `plotTracks` function defined in the **Gviz** package.
- The GRangesList class defined in the **GenomicRanges** package.
- The GAlignments and GAlignmentPairs classes defined in the **GenomicAlignments** package.
- The TxDb class defined in the **GenomicFeatures** package.

Examples

```
## ---------------------------------------------------------------------
## A. PLOT TRANSCRIPTS
## ---------------------------------------------------------------------
example(SplicingGraphs)  # create SplicingGraphs object 'sg'
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## The transcripts of a given gene can be extracted with [[. The result
## is an *unnamed* GRangesList object containing the exons grouped by
## transcript:
sg[["geneD"]]
plotTranscripts(sg[["geneD"]])  # requires the Gviz package

## The transcripts of all the genes can be extracted with unlist(). The
## result is a *named* GRangesList object containing the exons grouped
## by transcript. The names on the object are the gene ids:
ex_by_tx <- unlist(sg)
ex_by_tx
plotTranscripts(ex_by_tx)

## ---------------------------------------------------------------------
## B. PLOT TRANSCRIPTS AND READS
## ---------------------------------------------------------------------
gal <- readGAlignments(toy_reads_bam(), use.names=TRUE)
plotTranscripts(sg[["geneA"]], reads=gal)
plotTranscripts(ex_by_tx, reads=gal)
```

```
plotTranscripts(ex_by_tx, reads=gal, from=1, to=320)
plotTranscripts(ex_by_tx, reads=gal[21:26], from=1, to=320)
```

---

rsgedgesByGene-methods

Extract the reduced edges and their ranges from a SplicingGraphs object

---

Description

rsgedgesByGene and rsgedgesByTranscript are analog to sgedgesByGene and sgedgesByTranscript, but operate on the *reduced* splicing graphs, that is, the graphs in SplicingGraphs object x are reduced before the edges and their ranges are extracted. The reduced graphs are obtained by removing the uninformative nodes from it. See Details section below.

rsgedges extracts the edges of the reduced splicing graph of a given gene from a SplicingGraphs object.

rsgraph extracts the reduced splicing graph for a given gene from a SplicingGraphs object, and returns it as a plottable graph-like object.

Usage

```
rsgedgesByGene(x, with.hits.mcols=FALSE, keep.dup.edges=FALSE)

rsgedgesByTranscript(x, with.hits.mcols=FALSE)

rsgedges(x)

rsgraph(x, tx_id.as.edge.label=FALSE, as.igraph=FALSE)

## Related utility:
uninformativeSSids(x)
```

Arguments

x	A SplicingGraphs object. Must be of length 1 for rsgedges, rsgraph, and uninformativeSSids.
with.hits.mcols	Whether or not to include the *hits metadata columns* in the returned object. See ?countReads for more information.
keep.dup.edges	Not supported yet.
tx_id.as.edge.label	Whether or not to use the transcript ids as edge labels.
as.igraph	TODO

Details

TODO: Explain graph reduction.

Value

For `rsgedgesByGene`: A GRangesList object named with the gene ids and where the reduced splicing graph edges are grouped by gene.

TODO: Explain values returned by the other function.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see `?SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model (see
##    '?SplicingGraphs')
## ---------------------------------------------------------------------
example(SplicingGraphs)
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## ---------------------------------------------------------------------
## 2. rsgedgesByGene()
## ---------------------------------------------------------------------
edges_by_gene <- rsgedgesByGene(sg)
edges_by_gene
## 'edges_by_gene' has the length and names of 'sg', that is, the names
## on it are the gene ids and are guaranteed to be unique.

## Extract the reduced edges and their ranges for a given gene:
edges_by_gene[["geneA"]]
## Note that edge with global reduced edge id "geneA:1,2,4,5" is a mixed
## edge obtained by combining together edges "geneA:1,2" (exon),
## "geneA:2,4" (intron), and "geneA:4,5" (exon), during the graph
## reduction.

stopifnot(identical(edges_by_gene["geneB"], rsgedgesByGene(sg["geneB"])))

## ---------------------------------------------------------------------
## 3. sgedgesByTranscript()
## ---------------------------------------------------------------------
#edges_by_tx <- rsgedgesByTranscript(sg)  # not ready yet!
#edges_by_tx

## ---------------------------------------------------------------------
## 4. rsgedges(), rsgraph(), uninformativeSSids()
```

```
## ------------------------------------------------------------------------
plot(sgraph(sg["geneB"]))
uninformativeSSids(sg["geneB"])

plot(rsgraph(sg["geneB"]))
rsgedges(sg["geneB"])

## ------------------------------------------------------------------------
## 5. Sanity checks
## ------------------------------------------------------------------------
## TODO: Do the same kind of sanity checks that are done for sgedges()
## vs sgedgesByGene() vs sgedgesByTranscript() (in man page for sgedges).
```

---

sgedges-methods *Extract the edges (and nodes) of a splicing graph*

Description

sgedges (resp. sgnodes) extracts the edges (resp. the nodes) of the splicing graph of a given gene from a SplicingGraphs object.

Usage

```
sgedges(x, txweight=NULL, keep.dup.edges=FALSE)
sgnodes(x)
outdeg(x)
indeg(x)
```

Arguments

x	A SplicingGraphs object of length 1.
txweight	TODO
keep.dup.edges	If FALSE (the default), then edges with the same *global edge id* are merged into a single row. Use keep.dup.edges=TRUE if this merging should not be performed.

Details

TODO

Value

TODO

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model (see
##    '?SplicingGraphs')
## ---------------------------------------------------------------------
example(SplicingGraphs)  # create SplicingGraphs object 'sg'
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## ---------------------------------------------------------------------
## 2. Basic usage
## ---------------------------------------------------------------------
sgedges(sg["geneD"])
sgnodes(sg["geneD"])
outdeg(sg["geneD"])
indeg(sg["geneD"])

## ---------------------------------------------------------------------
## 3. Sanity checks
## ---------------------------------------------------------------------
check_way1_vs_way2 <- function(res1, res2)
{
    edges1 <- res1[res1$ex_or_in != "", ]  # remove artificial edges
    edges2 <- mcols(unlist(res2, use.names=FALSE))
    stopifnot(identical(edges1, edges2))
}

for (i in seq_along(sg)) {
    sgi <- sg[i]
    ## After removal of the artificial edges, the edges returned
    ## by 'sgedges()' should be the same as those returned
    ## by 'sgedgesByGene()' on a SplicingGraphs object of length 1.
    check_way1_vs_way2(
        sgedges(sgi),
        sgedgesByGene(sgi))
    ## After removal of the artificial edges, the edges returned
    ## by 'sgedges( , keep.dup.edges=TRUE)' should be the same as
    ## those returned by 'sgedgesByGene( , keep.dup.edges=TRUE)' or by
    ## 'sgedgesByTranscript()' on a SplicingGraphs object of length 1.
    res1 <- DataFrame(sgedges(sgi, keep.dup.edges=TRUE))
    check_way1_vs_way2(
        res1,
        sgedgesByGene(sgi, keep.dup.edges=TRUE))
    check_way1_vs_way2(
        res1,
```

```
        sgedgesByTranscript(sgi))
}
```

---

sgedgesByGene-methods *Extract the edges and their ranges from a SplicingGraphs object*

Description

sgedgesByGene and sgedgesByTranscript both extract the edges and their ranges of all the genes from a SplicingGraphs object. They return them in a GRangesList object named with the gene ids, and where the items are grouped by gene (for sgedgesByGene) or by transcript (for sgedgesByTranscript).

Alternatively, intronsByTranscript extracts the intronic edges and their ranges of all the genes from a SplicingGraphs object. It returns them in a GRangesList object named with the gene ids, and where the items are grouped by transcript.

Usage

```
sgedgesByGene(x, with.exon.mcols=FALSE, with.hits.mcols=FALSE,
                 keep.dup.edges=FALSE)

sgedgesByTranscript(x, with.exon.mcols=FALSE, with.hits.mcols=FALSE)

## S4 method for signature 'SplicingGraphs'
intronsByTranscript(x)
```

Arguments

x	A SplicingGraphs object.
with.exon.mcols	Whether or not to include the *exon metadata columns* in the returned object. Those columns are named: exon_id, exon_name, exon_rank, start_SSid, and end_SSid. They are set to NA for edges of type intron.
with.hits.mcols	Whether or not to include the *hits metadata columns* in the returned object. See ?countReads for more information.
keep.dup.edges	If FALSE (the default), then within each group of the returned object, edges with the same *global edge id* are merged into a single element. Use keep.dup.edges=TRUE if this merging should not be performed.

Value

A GRangesList object named with the gene ids and where the items are grouped by gene (for sgedgesByGene), or by transcript (for sgedgesByTranscript and intronsByTranscript). In the latter case (i.e. grouping by transcript), the names are not unique.

The items that are being grouped are the splicing graph edges of type exon and intron (no artificial edges) for `sgedgesByGene` and `sgedgesByTranscript`, and the introns for `intronsByTranscript`.

When the grouping is by transcript (i.e. for `sgedgesByTranscript` and `intronsByTranscript`, items are ordered by their position from 5' to 3'.

About duplicated edges: A given edge can typically be shared by more than 1 transcript within the same gene, therefore `sgedgesByTranscript` typically returns an object where the same *global edge id* shows up in more than 1 group. However, the same *global edge id* is never shared across genes. By default `sgedgesByGene` removes duplicated edges, unless `keep.dup.edges=TRUE` is used.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model (see
##    '?SplicingGraphs')
## ---------------------------------------------------------------------
example(SplicingGraphs)
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## ---------------------------------------------------------------------
## 2. sgedgesByGene()
## ---------------------------------------------------------------------
edges_by_gene <- sgedgesByGene(sg)
edges_by_gene
## 'edges_by_gene' has the length and names of 'sg', that is, the names
## on it are the gene ids and are guaranteed to be unique.

## Extract the edges and their ranges for a given gene:
edges_by_gene[["geneB"]]
## Note that edge with global edge id "geneB:3,4" is an intron that
## belongs to transcripts B1 and B2.

edges_by_gene0 <- sgedgesByGene(sg, keep.dup.edges=TRUE)
edges_by_gene0[["geneB"]]
## Note that edge "geneB:3,4" now shows up twice, once for transcript
## B1, and once for transcript B2.

## Keep the "exon metadata columns":
sgedgesByGene(sg, with.exon.mcols=TRUE)
## Note that those cols are set to NA for intronic edges.
```

```
## ---------------------------------------------------------------------
## 3. sgedgesByTranscript()
## ---------------------------------------------------------------------
edges_by_tx <- sgedgesByTranscript(sg)
edges_by_tx

## 'edges_by_tx' is typically longer than 'sg'.
## IMPORTANT NOTE: One caveat here is that the names on 'edges_by_tx'
## are the gene ids, not the transcript ids, and thus are typically NOT
## unique!

## Select elements of a given gene:
edges_by_tx["geneB"]  # not a good idea
edges_by_tx[names(edges_by_tx) %in% "geneB"]  # much better :-)
## Note that edge with global edge id "geneB:3,4" is an intron that
## belongs to transcripts B1 and B2.

## Keep the "exon metadata columns":
sgedgesByTranscript(sg, with.exon.mcols=TRUE)
## Note that those cols are set to NA for intronic edges.

## ---------------------------------------------------------------------
## 4. intronsByTranscript()
## ---------------------------------------------------------------------
in_by_tx <- intronsByTranscript(sg)
in_by_tx

## 'in_by_tx' has the length and names of 'edges_by_tx'. The same
## recommendation applies for selecting elements of a given set of
## genes:
in_by_tx[c("geneB", "geneD")]  # not a good idea
in_by_tx[names(in_by_tx) %in% c("geneB", "geneD")]  # much better :-)

## ---------------------------------------------------------------------
## 5. Comparing the outputs of unlist(), intronsByTranscript(), and
##    sgedgesByTranscript()
## ---------------------------------------------------------------------
ex_by_tx <- unlist(sg)
in_by_tx <- intronsByTranscript(sg)
edges_by_tx <- sgedgesByTranscript(sg)

## A sanity check:
stopifnot(identical(elementNROWS(in_by_tx) + 1L,
                    elementNROWS(ex_by_tx)))

## 'edges_by_tx' combines 'ex_by_tx' and 'in_by_tx' in a single
## GRangesList object. Sanity check:
stopifnot(identical(elementNROWS(edges_by_tx),
                    elementNROWS(ex_by_tx) + elementNROWS(in_by_tx)))
```

sgraph-methods *Extract a splicing graph as a plottable graph-like object*

Description

Extract the splicing graph for a given gene from a SplicingGraphs object and return it as a plottable graph-like object.

Usage

```
sgraph(x, keep.dup.edges=FALSE, tx_id.as.edge.label=FALSE, as.igraph=FALSE)

## PLotting:

## S4 method for signature 'SplicingGraphs,missing'
plot(x, y, ...)
slideshow(x)
```

Arguments

x	TODO
keep.dup.edges	If FALSE (the default), then edges with the same *global edge id* are merged together. Use keep.dup.edges=TRUE if this merging should not be performed.
tx_id.as.edge.label	Whether or not to use the transcript ids as edge labels.
as.igraph	TODO
y	TODO
...	Additional arguments. plot passes these to plot,Ragraph,ANY-method for use in, e.g., adding a main title to the plot.

Details

TODO

Value

TODO

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Examples

```
example(SplicingGraphs)  # create SplicingGraphs object 'sg'
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

graphA <- sgraph(sg["geneA"], tx_id.as.edge.label=TRUE)

if (interactive()) {
  ## Edges are labeled with the transcript ids (or names), in blue.
  ## The orange arrows are edges corresponding to exons:
  plot(graphA)

  ## Note that plot() works directly on a SplicingGraphs object of
  ## length 1:
  plot(sg["geneB"])

  ## Slideshow of the graphs:
  slideshow(sg)
}
```

SplicingGraphs-class *SplicingGraphs objects*

Description

The SplicingGraphs class is a container for storing splicing graphs together with the gene model that they are based on.

Usage

```
## Constructor:

SplicingGraphs(x, grouping=NULL, min.ntx=2, max.ntx=NA, check.introns=TRUE)

## SplicingGraphs basic API:

## S4 method for signature 'SplicingGraphs'
length(x)

## S4 method for signature 'SplicingGraphs'
names(x)

## S4 method for signature 'SplicingGraphs'
seqnames(x)

## S4 method for signature 'SplicingGraphs'
```

```
strand(x)

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

## S4 method for signature 'SplicingGraphs,ANY,ANY'
x[[i, j, ...]]

## S4 method for signature 'SplicingGraphs'
elementNROWS(x)

## S4 method for signature 'SplicingGraphs'
unlist(x, recursive=TRUE, use.names=TRUE)

## S4 method for signature 'SplicingGraphs'
seqinfo(x)
```

Arguments

x	For SplicingGraphs: A GRangesList object containing the exons of one or more genes grouped by transcript. Alternatively, x can be a TxDb object. See Details section below. For the methods in the SplicingGraphs basic API: A SplicingGraphs object.
grouping	An optional object that represents the grouping by gene of the top-level elements (i.e. the transcripts) in x. See Details section below.
min.ntx, max.ntx	Single integers (or NA for max.ntx) specifying the minimum and maximum number of transcripts a gene must have to be considered for inclusion in the object returned by SplicingGraphs. A value of NA for max.ntx means no maximum.
check.introns	If TRUE, SplicingGraphs checks that, within each transcript, exons are ordered from 5' to 3' with gaps of at least 1 nucleotide between them.
i, j, ..., drop	A SplicingGraphs object is a list-like object and therefore it can be subsetted like a list. When subsetting with [, the result is another SplicingGraphs object containing only the selected genes. When subsetting with [[, the result is an *unnamed* GRangesList object containing the exons grouped by transcript. Like for list, subsetting only accepts 1 argument (i). The drop argument is ignored and trying to pass any additional argument (to j or in ...) will raise an error.
recursive, use.names	A SplicingGraphs object is a list-like object and therefore it can be unlisted with unlist. The result is a GRangesList object containing the exons grouped by transcript. By default this object has names on it, and the names are the gene ids. Note that because each element in this object represents a transcript (and not a gene), the names are not unique. If use.names=FALSE is used, the result has no names on it. The recursive agument is ignored.

Details

The Splicing graph theory only applies to genes that have all the exons of all their transcripts on the same chromosome and strand. In particular, in its current form, the splicing graph theory cannot describe trans-splicing events. The `SplicingGraphs` constructor will reject genes that do not satisfy this.

The first argument of the `SplicingGraphs` constructor, `x`, can be either a GRangesList object or a TxDb object.

When `x` is a GRangesList object, it must contain the exons of one or more genes grouped by transcripts. More precisely, each top-level element in `x` must contain the genomic ranges of the exons for a particular transcript. Typically `x` will be obtained from a TxDb object `txdb` with `exonsBy(txdb, by="tx", use.names=TRUE)`.

`grouping` is an optional argument that is only supported when `x` is a GRangesList object. It represents the grouping by gene of the top-level elements (i.e. the transcripts) in GRangesList object `x`. It can be either:

- Missing (i.e. `NULL`). In that case, all the transcripts in `x` are considered to belong to the same gene and the SplicingGraphs object returned by `SplicingGraphs` will be unnamed.
- A list of integer or character vectors, or an IntegerList, or a CharacterList object, of length the number of genes to process, and where `grouping[[i]]` is a vector of valid subscripts in `x` pointing to all the transcripts of the i-th gene.
- A factor, character vector, or integer vector, of the same length as `x` and 1 level per gene.
- A named GRangesList object containing transcripts grouped by genes i.e. each top-level element in `grouping` contains the genomic ranges of the transcripts for a particular gene. In that case, the grouping is inferred from the tx_id (or alternatively tx_name) metadata column of `unlist(grouping)` and all the values in that column must be in `names(x)`. If `x` was obtained with `exonsBy(txdb, by="tx", use.names=TRUE)`, then the GRangesList object used for grouping would typically be obtained with `transcriptsBy(txdb, by="gene")`.
- A data.frame or DataFrame with 2 character vector columns: a gene_id column (factor, character vector, or integer vector), and a tx_id (or alternatively tx_name) column. In that case, `x` must be named and all the values in the tx_id (or tx_name) column must be in `names(x)`.

Value

For `SplicingGraphs`: a SplicingGraphs object with 1 element per gene.

For `length`: the number of genes in `x`, which is also the number of splicing graphs in `x`.

For `names`: the gene ids. Note that the names on a SplicingGraphs object are always unique and cannot be modified.

For `seqnames`: a named factor of the length of `x` containing the name of the chromosome for each gene.

For `strand`: a named factor of the length of `x` containing the strand for each gene.

For `elementNROWS`: the number of transcripts per gene.

For `seqinfo`: the seqinfo of the GRangesList or TxDb object that was used to construct the SplicingGraphs object.

Author(s)

H. Pagès

References

Heber, S., Alekseyev, M., Sze, S., Tang, H., and Pevzner, P. A. *Splicing graphs and EST assembly problem* Bioinformatics Date: Jul 2002 Vol: 18 Pages: S181-S188

Sammeth, M. (2009) *Complete Alternative Splicing Events Are Bubbles in Splicing Graphs* J. Comput. Biol. Date: Aug 2009 Vol: 16 Pages: 1117-1140

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Other topics related to this man page and documented in other packages:

- The exonsBy and transcriptsBy functions, and the TxDb class, defined in the **GenomicFeatures** package.
- The GRangesList class defined in the **GenomicRanges** package.
- The IntegerList and CharacterList classes defined in the **IRanges** package.
- The DataFrame class defined in the **S4Vectors** package.

Examples

```
## ---------------------------------------------------------------------
## 1. Load a toy gene model as a TxDb object
## ---------------------------------------------------------------------

library(txdbmaker)
suppressWarnings(
  toy_genes_txdb <- makeTxDbFromGFF(toy_genes_gff())
)

## ---------------------------------------------------------------------
## 2. Compute all the splicing graphs (1 graph per gene) and return them
##    in a SplicingGraphs object
## ---------------------------------------------------------------------

## Extract the exons grouped by transcript:
ex_by_tx <- exonsBy(toy_genes_txdb, by="tx", use.names=TRUE)

## Extract the transcripts grouped by gene:
tx_by_gn <- transcriptsBy(toy_genes_txdb, by="gene")

sg <- SplicingGraphs(ex_by_tx, tx_by_gn)
sg

## Alternatively 'sg' can be constructed directly from the TxDb
## object:
sg2 <- SplicingGraphs(toy_genes_txdb)  # same as 'sg'
```

```
sg2

## Note that because SplicingGraphs objects have a slot that is an
## environment (for caching the bubbles), they cannot be compared with
## 'identical()' (will always return FALSE). 'all.equal()' should be
## used instead:
stopifnot(isTRUE(all.equal(sg2, sg)))

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids:
length(sg)
names(sg)

## ---------------------------------------------------------------------
## 3. Basic manipulation of a SplicingGraphs object
## ---------------------------------------------------------------------

## Basic accessors:
seqnames(sg)
strand(sg)
seqinfo(sg)

## Number of transcripts per gene:
elementNROWS(sg)

## The transcripts of a given gene can be extracted with [[. The result
## is an *unnamed* GRangesList object containing the exons grouped by
## transcript:
sg[["geneD"]]

## See '?plotTranscripts' for how to plot those transcripts.

## The transcripts of all the genes can be extracted with unlist(). The
## result is a *named* GRangesList object containing the exons grouped
## by transcript. The names on the object are the gene ids:
ex_by_tx <- unlist(sg)
ex_by_tx
```

`toy_data` *Little helpers for quick access to the toy data included in the* **SplicingGraphs** *package*

Description

TODO

Usage

```
toy_genes_gff()
toy_reads_sam()
toy_reads_bam()
toy_overlaps()
```

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Other topics related to this man page and documented in other packages:

- The GRangesList class defined in the **GenomicRanges** package.
- The GAlignments and GAlignmentPairs classes defined in the **GenomicAlignments** package.
- The makeTxDbFromGFF function in the **txdbmaker** package.
- The TxDb class defined in the **GenomicFeatures** package.

Examples

```
## ---------------------------------------------------------------------
## A. LOAD THE TOY GENE MODEL AS A TxDb OBJECT AND PLOT IT
## ---------------------------------------------------------------------
toy_genes_gff()

## Note that you can display the content of the file with:
cat(readLines(toy_genes_gff()), sep="\n")

library(txdbmaker)
suppressWarnings(
  txdb <- makeTxDbFromGFF(toy_genes_gff())
)

## Plot all the transcripts in the gene model:
plotTranscripts(txdb)

## ---------------------------------------------------------------------
## B. LOAD THE TOY READS AS A GAlignments OBJECT AND PLOT THEM
## ---------------------------------------------------------------------
## The reads are single-end reads. They are assumed to come from an
## RNA-seq experiment and to have been aligned to the exact same
## reference genome that the above toy gene model is based on.
toy_reads_sam()
toy_reads_bam()
gal <- readGAlignments(toy_reads_bam(), use.names=TRUE)

plotTranscripts(txdb, reads=gal)
plotTranscripts(txdb, reads=gal, from=1, to=320)

## ---------------------------------------------------------------------
## C. FIND THE OVERLAPS BETWEEN THE TOY READS AND THE TOY GENE MODEL
## ---------------------------------------------------------------------
grl <- grglist(gal, order.as.in.query=TRUE)
ex_by_tx <- exonsBy(txdb, by="tx", use.names=TRUE)
```

```
## Most of the times the RNA-seq protocol is unstranded so the strand
## reported in the BAM file for each alignment is meaningless. Thus we
## should call findOverlaps() with 'ignore.strand=TRUE':
ov0 <- findOverlaps(grl, ex_by_tx, ignore.strand=TRUE)

## Sort and put the overlaps in a data.frame to make them easier to
## read:
ov0 <- sort(ov0)
df0 <- data.frame(QNAME=names(grl)[queryHits(ov0)],
                  tx_id=names(ex_by_tx)[subjectHits(ov0)],
                  stringsAsFactors=FALSE)
head(df0)

## These overlaps have been manually checked and included in the
## SplicingGraphs package. They can be loaded with the toy_overlaps()
## helper:
toy_ov <- toy_overlaps()
head(toy_ov)
stopifnot(identical(df0, toy_ov[ , 1:2]))

## ---------------------------------------------------------------------
## D. DETECT THE OVERLAPS THAT ARE COMPATIBLE WITH THE GENE MODEL
## ---------------------------------------------------------------------
## First we encode the overlaps:
ovenc0 <- encodeOverlaps(grl, ex_by_tx, hits=ov0,
                         flip.query.if.wrong.strand=TRUE)
ovenc0

## Each encoding tells us whether the corresponding overlap is
## compatible or not with the gene model:
ov0_is_comp <- isCompatibleWithSplicing(ovenc0)
head(ov0_is_comp)

## Overlap compatibility has also been manually checked and included in
## the table returned by toy_overlaps():
stopifnot(identical(ov0_is_comp, toy_ov[ , 3]))
```

TSPCsg *TSPC splicing graphs*

Description

TODO

Examples

```
## Load SplicingGraphs object 'TSPCsg':
filepath <- system.file("extdata", "TSPCsg.rda", package="SplicingGraphs")
load(filepath)
```

```
TSPCsg

## 'TSPCsg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(TSPCsg)

## 1 splicing graph per gene. (Note that gene MUC16 was dropped
## because transcripts T-4 and T-5 in this gene both have their
## 2nd exon *inside* their 3rd exon. Splicing graph theory doesn't
## apply in that case.)

## Extract the edges of a given graph:
TSPCsgedges <- sgedges(TSPCsg["LGSN"])
TSPCsgedges

## Plot the graph for a given gene:
plot(TSPCsg["LGSN"])  # or 'plot(sgraph(TSPCsgedges))'

## The reads from all samples have been assigned to 'TSPCsg'.
## Use countReads() to summarize by splicing graph edge:
counts <- countReads(TSPCsg)
dim(counts)
counts[ , 1:5]

## You can subset 'TSPCsg' by 1 or more gene ids before calling
## countReads() in order to summarize only for those genes:
DAPL1counts <- countReads(TSPCsg["DAPL1"])
dim(DAPL1counts)
DAPL1counts[ , 1:5]

## Use 'by="rsgedge"' to summarize by *reduced* splicing graph edge:
DAPL1counts2 <- countReads(TSPCsg["DAPL1"], by="rsgedge")
dim(DAPL1counts2)
DAPL1counts2[ , 1:5]

## No reads assigned to genes KIAA0319L or TREM2 because no
## BAM files were provided for those genes:
KIAA0319Lcounts <- countReads(TSPCsg["KIAA0319L"])
KIAA0319Lcountsums <- sapply(KIAA0319Lcounts[ , -(1:2)], sum)
stopifnot(all(KIAA0319Lcountsums == 0))

TREM2counts <- countReads(TSPCsg["TREM2"])
TREM2countsums <- sapply(TREM2counts[ , -(1:2)], sum)
stopifnot(all(TREM2countsums == 0))

## Plot all the splicing graphs:
slideshow(TSPCsg)
```

---

txpath-methods *Extract the transcript paths of a splicing graph*

Description

txpath extracts the transcript paths of the splicing graph of a given gene from a SplicingGraphs object.

Usage

```
txpath(x, as.matrix=FALSE)

## Related utility:
txweight(x)
txweight(x) <- value
```

Arguments

x	A SplicingGraphs object of length 1 or a GRangesList object for txpath. A SplicingGraphs object for txweight.
as.matrix	TODO
value	A numeric vector containing the weights to assign to each transcript in x.

Details

TODO

Value

A named list-like object with one list element per transcript in the gene. Each list element is an integer vector that describes the *path* of the transcript i.e. the *Splicing Site ids* that it goes thru.

Author(s)

H. Pagès

See Also

This man page is part of the **SplicingGraphs** package. Please see ?`SplicingGraphs-package` for an overview of the package and for an index of its man pages.

Other topics related to this man page and documented in other packages:

- The GRangesList class defined in the **GenomicRanges** package.
- The GAlignments and GAlignmentPairs classes defined in the **GenomicAlignments** package.
- findOverlaps-methods in the **GenomicRanges** package.
- encodeOverlaps-methods in the **GenomicAlignments** package.
- The ScanBamParam function defined in the **Rsamtools** package.

Examples

```
## ---------------------------------------------------------------------
## 1. Make SplicingGraphs object 'sg' from toy gene model (see
##    '?SplicingGraphs')
## ---------------------------------------------------------------------
example(SplicingGraphs)
sg

## 'sg' has 1 element per gene and 'names(sg)' gives the gene ids.
names(sg)

## ---------------------------------------------------------------------
## 2. txpath()
## ---------------------------------------------------------------------
## Note that the list elements in the returned IntegerList object
## always consist of an even number of Splicing Site ids in ascending
## order.
txpath(sg["geneB"])
txpath(sg["geneD"])
strand(sg)

txpath(sg["geneD"], as.matrix=TRUE)  # splicing matrix

## ---------------------------------------------------------------------
## 3. txweight()
## ---------------------------------------------------------------------
txweight(sg)
plot(sg["geneD"])

txweight(sg) <- 5
txweight(sg)
plot(sg["geneD"])  # FIXME: Edges not rendered with correct width!
plot(sgraph(sg["geneD"], as.igraph=TRUE))  # need to use this for now

txweight(sg)[8:11] <- 5 * (4:1)
txweight(sg)
plot(sgraph(sg["geneD"], tx_id.as.edge.label=TRUE, as.igraph=TRUE))

## ---------------------------------------------------------------------
## 4. An advanced example
## ---------------------------------------------------------------------
## [TODO: Counting "unambiguous compatible hits" per transcript should be
##  supported by countReads(). Simplify the code below when countReads()
##  supports this.]
## Here we show how to find "unambiguous compatible hits" between a set
## of RNA-seq reads and a set of transcripts, that is, hits that are
## compatible with the splicing of exactly 1 transcript. Then we set the
## transcript weights based on the number of unambiguous compatible hits
## they received and we finally plot some splicing graphs that show
## the weighted transcripts.
## Note that the code we use to find the unambiguous compatible hits
## uses findOverlaps() and encodeOverlaps() defined in the IRanges and
```

```
## GenomicRanges packages. It only requires that the transcripts are
## represented as a GRangesList object and the reads as a GAlignments
## (single-end) or GAlignmentPairs (paired-end) object, and therefore is
## not specific to SplicingGraphs.

## First we load toy reads (single-end) from a BAM file. We filter out
## secondary alignments, reads not passing quality controls, and PCR or
## optical duplicates (see ?scanBamFlag in the Rsamtools package for
## more information):
flag0 <- scanBamFlag(isSecondaryAlignment=FALSE,
                     isNotPassingQualityControls=FALSE,
                     isDuplicate=FALSE)
param0 <- ScanBamParam(flag=flag0)
gal <- readGAlignments(toy_reads_bam(), use.names=TRUE, param=param0)
gal

## Put the reads in a GRangesList object:
grl <- grglist(gal, order.as.in.query=TRUE)

## Put the transcripts in a GRangesList object (made of exons grouped
## by transcript):
ex_by_tx <- unlist(sg)

## Most of the times the RNA-seq protocol is unstranded so the strand
## reported in the BAM file (and propagated to 'grl') for each alignment
## is meaningless. Thus we should call findOverlaps() with
## 'ignore.strand=TRUE':
ov0 <- findOverlaps(grl, ex_by_tx, ignore.strand=TRUE)

## Encode the overlaps (this compare the fragmentation of the reads with
## the splicing of the transcripts):
ovenc0 <- encodeOverlaps(grl, ex_by_tx, hits=ov0,
                         flip.query.if.wrong.strand=TRUE)
ov0_is_compat <- isCompatibleWithSplicing(ovenc0)

## Keep compatible overlaps only:
ov1 <- ov0[ov0_is_compat]

## Only keep overlaps that are compatible with exactly 1 transcript:
ov2 <- ov1[queryHits(ov1) %in% which(countQueryHits(ov1) == 1L)]
nhit_per_tx <- countSubjectHits(ov2)
names(nhit_per_tx) <- names(txweight(sg))
nhit_per_tx

txweight(sg) <- 2 * nhit_per_tx
plot(sgraph(sg["geneA"], tx_id.as.edge.label=TRUE, as.igraph=TRUE))
plot(sgraph(sg["geneB"], tx_id.as.edge.label=TRUE, as.igraph=TRUE))
```

Index

∗ **package**
SplicingGraphs-package, 2
[,SplicingGraphs,ANY,ANY,ANY-method *(SplicingGraphs-class)*, 21
[[,SplicingGraphs,ANY,ANY-method *(SplicingGraphs-class)*, 21

ASCODE2DESC *(bubbles-methods)*, 7
assignReads, *4*, 4, *9*

bubbles, *3*
bubbles *(bubbles-methods)*, 7
bubbles,ANY-method *(bubbles-methods)*, 7
bubbles,IntegerList-method *(bubbles-methods)*, 7
bubbles,SplicingGraphs-method *(bubbles-methods)*, 7
bubbles-methods, 7

CharacterList, *23, 24*
class:GeneModel *(SplicingGraphs-class)*, 21
class:SplicingGraphs *(SplicingGraphs-class)*, 21
countReads, *4*, *13*, *17*
countReads *(countReads-methods)*, 8
countReads,SplicingGraphs-method *(countReads-methods)*, 8
countReads-methods, 8

DataFrame, *9*, *23, 24*

elementNROWS,SplicingGraphs-method *(SplicingGraphs-class)*, 21
encodeOverlaps-methods, *29*
exonsBy, *23, 24*

findOverlaps-methods, *29*

GAlignmentPairs, *4, 5*, *11, 12*, *26*, *29*
GAlignments, *4, 5*, *11, 12*, *26*, *29*
GeneModel-class *(SplicingGraphs-class)*, 21
GRangesList, *4, 5*, *11, 12*, *14*, *17*, *22–24*, *26*, *29*

indeg *(sgedges-methods)*, 15
indeg,ANY-method *(sgedges-methods)*, 15
indeg,DataFrame-method *(sgedges-methods)*, 15
IntegerList, *23, 24*
intronsByTranscript,SplicingGraphs-method *(sgedgesByGene-methods)*, 17
isCircular<-,GeneModel-method *(SplicingGraphs-class)*, 21
isCircular<-,SplicingGraphs-method *(SplicingGraphs-class)*, 21

length,SplicingGraphs-method *(SplicingGraphs-class)*, 21
lengths,SplicingGraphs-method *(SplicingGraphs-class)*, 21

makeTxDbFromGFF, *26*

names,SplicingGraphs-method *(SplicingGraphs-class)*, 21

outdeg *(sgedges-methods)*, 15
outdeg,ANY-method *(sgedges-methods)*, 15
outdeg,DataFrame-method *(sgedges-methods)*, 15

plot,SplicingGraphs,missing-method *(sgraph-methods)*, 20
plotTracks, *12*
plotTranscripts, *3*
plotTranscripts *(plotTranscripts-methods)*, 11
plotTranscripts,GRangesList-method *(plotTranscripts-methods)*, 11

plotTranscripts,SplicingGraphs-method
(plotTranscripts-methods), 11
plotTranscripts,TxDb-method
(plotTranscripts-methods), 11
plotTranscripts-methods, 11

quickBamFlagSummary, *5*

readGAlignmentPairs, *5*
readGAlignments, *5*
removeReads (assignReads), 4
rsgedges (rsgedgesByGene-methods), 13
rsgedgesByGene, *3*
rsgedgesByGene
(rsgedgesByGene-methods), 13
rsgedgesByGene,SplicingGraphs-method
(rsgedgesByGene-methods), 13
rsgedgesByGene-methods, 13
rsgedgesByTranscript
(rsgedgesByGene-methods), 13
rsgedgesByTranscript,SplicingGraphs-method
(rsgedgesByGene-methods), 13
rsgraph (rsgedgesByGene-methods), 13

scanBamFlag, *5*
ScanBamParam, *5*, *29*
seqinfo,GeneModel-method
(SplicingGraphs-class), 21
seqinfo,SplicingGraphs-method
(SplicingGraphs-class), 21
seqnames,GeneModel-method
(SplicingGraphs-class), 21
seqnames,SplicingGraphs-method
(SplicingGraphs-class), 21
sgedges, *3*
sgedges (sgedges-methods), 15
sgedges,SplicingGraphs-method
(sgedges-methods), 15
sgedges-methods, 15
sgedges2 (rsgedgesByGene-methods), 13
sgedgesByGene, *3*, *13*
sgedgesByGene (sgedgesByGene-methods),
17
sgedgesByGene,SplicingGraphs-method
(sgedgesByGene-methods), 17
sgedgesByGene-methods, 17
sgedgesByTranscript, *13*
sgedgesByTranscript
(sgedgesByGene-methods), 17
sgedgesByTranscript,SplicingGraphs-method
(sgedgesByGene-methods), 17
sgnodes (sgedges-methods), 15
sgnodes,data.frame-method
(sgedges-methods), 15
sgnodes,DataFrame-method
(sgedges-methods), 15
sgnodes,IntegerList-method
(sgedges-methods), 15
sgnodes,SplicingGraphs-method
(sgedges-methods), 15
sgraph, *3*
sgraph (sgraph-methods), 20
sgraph,ANY-method (sgraph-methods), 20
sgraph,data.frame-method
(sgraph-methods), 20
sgraph,DataFrame-method
(sgraph-methods), 20
sgraph,igraph-method (sgraph-methods),
20
sgraph-methods, 20
sgraph2 (rsgedgesByGene-methods), 13
show,SplicingGraphs-method
(SplicingGraphs-class), 21
slideshow (sgraph-methods), 20
SplicingGraphs, *3*, *4*, *7*, *8*, *11*, *13*, *15*, *17*, *20*,
29
SplicingGraphs (SplicingGraphs-class),
21
SplicingGraphs,GRangesList-method
(SplicingGraphs-class), 21
SplicingGraphs,TxDb-method
(SplicingGraphs-class), 21
SplicingGraphs-class, 21
SplicingGraphs-package, 2
strand,GeneModel-method
(SplicingGraphs-class), 21
strand,SplicingGraphs-method
(SplicingGraphs-class), 21

toy_data, 25
toy_genes_gff, *4*
toy_genes_gff (toy_data), 25
toy_overlaps (toy_data), 25
toy_reads_bam (toy_data), 25
toy_reads_sam (toy_data), 25
transcriptsBy, *23*, *24*
TSPC (TSPCsg), 27
TSPCsg, 27

TxDb, *11*, *12*, *22–24*, *26*
txpath, *3*
txpath *(*txpath-methods*)*, 28
txpath,GRangesList-method
*(*txpath-methods*)*, 28
txpath,SplicingGraphs-method
*(*txpath-methods*)*, 28
txpath-methods, 28
txweight *(*txpath-methods*)*, 28
txweight,SplicingGraphs-method
*(*txpath-methods*)*, 28
txweight<- *(*txpath-methods*)*, 28
txweight<-,SplicingGraphs-method
*(*txpath-methods*)*, 28

uninformativeSSids
*(*rsgedgesByGene-methods*)*, 13
uninformativeSSids,ANY-method
*(*rsgedgesByGene-methods*)*, 13
uninformativeSSids,DataFrame-method
*(*rsgedgesByGene-methods*)*, 13
unlist,SplicingGraphs-method
*(*SplicingGraphs-class*)*, 21
updateObject,SplicingGraphs-method
*(*SplicingGraphs-class*)*, 21