Package 'splicelogic'

September 7, 2026

Title splicelogic: differential transcripts to splice events

Version 1.1.4

Description Translate differential transcript usage results into discrete splice events.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

URL https://github.com/thelovelab/splicelogic,
https://thelovelab.github.io/splicelogic

biocViews AlternativeSplicing, DifferentialSplicing, Transcriptomics, RNASeq, LongRead, Annotation, FunctionalGenomics

BugReports https://github.com/thelovelab/splicelogic/issues

Depends R (>= 4.5.0)

Imports dplyr, magrittr, GenomicRanges, plyranges, tibble, IRanges, S4Vectors, rlang, methods, stats

Suggests knitr, rmarkdown, testthat (>= 3.0.0), readr, wiggleplotr, GenomicFeatures, AnnotationHub, ggplot2, AnnotationDbi, Seqinfo, BSgenome, Biostrings, BSgenome.Hsapiens.UCSC.hg38

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

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

git_branch devel

git_last_commit b78a80f

git_last_commit_date 2026-09-04

Repository Bioconductor 3.24

Date/Publication 2026-09-06

Author Beatriz Campillo [aut, cre] (ORCID: <https://orcid.org/0000-0001-7323-9125>), Michael Love [aut] (ORCID: <https://orcid.org/0000-0001-8401-0545>), NIH NHGRI [fnd], Wellcome Trust [fnd]

Maintainer Beatriz Campillo <beatrizcampillo29@gmail.com>

Contents

splicelogic-package 2
create_mock_data 3
find_events 4
generate_events 7
get_seq 8
prepare_exons 9
prepare_exons_by_partition 11
preprocess 12

Index **13**

`splicelogic-package` *splicelogic: differential transcripts to splice events*

Description

For more details on the features of splicelogic, read the vignette: `browseVignettes(package = "splicelogic")`

Author(s)

Maintainer: Beatriz Campillo <beatrizcampillo29@gmail.com> (ORCID)

Authors:

- Michael Love <michaelisaiahlove@gmail.com> (ORCID)

Other contributors:

- NIH NHGRI [funder]
- Wellcome Trust [funder]

See Also

Useful links:

- https://github.com/thelovelab/splicelogic
- https://thelovelab.github.io/splicelogic
- Report bugs at https://github.com/thelovelab/splicelogic/issues

`create_mock_data` *Create mock GRanges data for splicing event testing*

Description

Create mock GRanges data for splicing event testing

Usage

```
create_mock_data(
  n_genes = 1,
  n_tx_per_gene = 2,
  n_exons_per_tx = 5,
  coef_range = c(-1, 1),
  strand = c("+", "-", "*")
)
```

Arguments

`n_genes`	Number of genes to simulate
`n_tx_per_gene`	Number of transcripts per gene. Use 2 or more: the first transcript of each gene is given a negative estimate and the second a positive one, so a single transcript per gene leaves the set all-negative and the `generate_*()` helpers with nothing to modify.
`n_exons_per_tx`	Number of exons per transcript
`coef_range`	Range of coefficient values to sample from
`strand`	Strand to place every transcript on: `"+"` (default), `"-"`, or `"*"`. On `"-"` the exon ranks are reversed, so exon_rank 1 is the rightmost exon in genomic coordinates.

Value

A GRanges object with simulated transcripts and exons

Examples

```
# create mock data with 2 genes, 4 transcripts
# per gene, and 4 exons per transcript
gr <- create_mock_data(n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4)

# the same, on the minus strand
gr_minus <- create_mock_data(n_genes = 2, n_exons_per_tx = 4, strand = "-")
```

`find_events` *Find splice events from annotated exons*

Description

Functions to find different types of alternative splicing events from preprocessed GRanges exon data. Events include skipped exon (se), included exon (ie), mutatualy exclusive exons (mxe), retained intron (ri), alternative 5' and 3' splice sites (a5ss / a3ss), and alternative transcription start and end sites (atss / ates).

`find_a5ss()` / `find_a3ss()` do not require either boundary of an exon to be shared with its partner, so an exon that moved both boundaries is reported as an a5ss *and* an a3ss. A first exon has no acceptor and a last exon has no donor, so those boundaries never produce an a3ss / a5ss; `find_atss()` / `find_ates()` report them instead by comparing where each transcript begins and ends. `exon_rank` runs in transcription order, so this holds on both strands.

Usage

```
find_se(gr, type = c("boundary", "over", "in"), inverse = FALSE)

find_skipped_exons(gr, type = c("boundary", "over", "in"), inverse = FALSE)

find_ie(gr, type = c("boundary", "over", "in"))

find_included_exons(gr, type = c("boundary", "over", "in"))

find_mxe(gr, type = c("boundary", "in", "over"))

find_mutually_exclusive_exons(gr, type = c("boundary", "in", "over"))

find_ri(gr)

find_retained_introns(gr)

find_a5ss(gr)

find_alternative_5_prime_splice_sites(gr)

find_a3ss(gr)

find_alternative_3_prime_splice_sites(gr)

find_atss(gr)

find_alternative_transcription_start_sites(gr)

find_ates(gr)

find_alternative_transcription_end_sites(gr)

find_all_events(gr, type = c("boundary", "over", "in"), verbose = TRUE)
```

Arguments

gr	A GRanges object with exon annotations, including 'tx_id', 'exon', and 'coef_col' metadata columns and preprocessed with preprocess().
type	How an exon flanking a candidate is matched to an exon of the partner transcript. One of:
	"boundary" (default) the two exons overlap *and* share a start or an end coordinate, so they may still differ in length at the other end.
	"over" any overlap, with no coordinate in common required. The most permissive setting.
	"in" the two exons are identical (same start and same end). The strictest setting.
inverse	If TRUE, identifies included exons instead of skipped exons.
verbose	If TRUE, prints progress messages. Default TRUE.

Value

A GRanges object with the detected exon ranges and the following additional metadata columns:

event_type The type of splicing event detected (e.g. "se", "ie", "mxe", "ri", "a5ss", "a3ss", "atss", "ates").

event_tx_id Transcript ID of the paired transcript involved in the event.

event_estimate DTU coefficient of the paired transcript.

event_<col> One column per name in metadata(gr)$additional_columns, prefixed with event_, carrying the corresponding value from the paired transcript.

find_se(): skipped exons

find_ie(): included exons

find_mxe(): mutually exclusive exons

find_ri(): retained introns

find_a5ss(): alternative 5' splice sites

find_a3ss(): alternative 3' splice sites

find_atss(): alternative transcription start sites — first exons that begin transcription at a different coordinate

find_ates(): alternative transcription end sites — last exons that end transcription at a different coordinate

find_all_events(): all detected events

Examples

```
# make some mock data and run the function
gr <- create_mock_data(n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4) |>
  preprocess(coef_col = "estimate") |>
  generate_se(n_events = 1)

# this should find the skipped exon events we generated
find_se(gr, type = "boundary")

find_ie(gr, type = "boundary")
```

```
# detect mutually exclusive exons
gr_mx <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
) |>
  preprocess(coef_col = "estimate") |>
  generate_mxe(n_events = 1)

find_mxe(gr_mx, type = "boundary")


# detect retained introns
gr_ri <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
) |>
  preprocess(coef_col = "estimate") |>
  generate_ri(n_events = 1)

find_ri(gr_ri)


# detect alternative 5' splice sites
gr_a5 <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
) |>
  preprocess(coef_col = "estimate") |>
  generate_a5ss(n_events = 1)

find_a5ss(gr_a5)


# detect alternative 3' splice sites
gr_a3 <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
) |>
  preprocess(coef_col = "estimate") |>
  generate_a3ss(n_events = 1)
find_a3ss(gr_a3)


gr_tss <- data.frame(
  seqnames = "chr1",
  start = c(1, 21, 41, 5, 21, 41),
  end = c(10, 30, 50, 10, 30, 50),
  strand = "+",
  gene_id = "g1",
  tx_id = rep(c("down", "up"), each = 3),
  exon_rank = rep(seq_len(3), 2),
  estimate = rep(c(-1, 1), each = 3)
) |>
  plyranges::as_granges() |>
  preprocess(coef_col = "estimate")

find_atss(gr_tss)


# detect all event types at once
gr_all <- create_mock_data(
```

```
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
) |>
  preprocess(coef_col = "estimate") |>
  generate_se(n_events = 1)

find_all_events(gr_all, type = "boundary", verbose = FALSE)
```

`generate_events` *Generate mock splice events in a GRanges object*

Description

Functions to introduce specific types of alternative splicing events into mock GRanges data for testing purposes.

Usage

```
generate_se(gr, n_events = 1)

generate_mxe(gr, n_events = 1)

generate_ri(gr, n_events = 1)

generate_a5ss(gr, n_events = 1)

generate_a3ss(gr, n_events = 1)

generate_atss(gr, n_events = 1, mode = c("shift", "drop"))

generate_ates(gr, n_events = 1, mode = c("shift", "drop"))
```

Arguments

`gr`	A GRanges object with metadata columns: 'exon_rank', 'gene_id', 'tx_id', and 'estimate'.
`n_events`	Number of events to generate
`mode`	How to move the site: `"shift"` moves the terminal exon's outer boundary (so it still overlaps its partner), `"drop"` removes the terminal exon and re-ranks (so the new terminal exon does not overlap its partner at all).

Value

`generate_se()`: A GRanges object with skipped exon events introduced

`generate_mxe()`: A GRanges object with mutually exclusive exon events introduced

`generate_ri()`: A GRanges object with retained intron events introduced

`generate_a5ss()`: A GRanges object with alternative 5' splice site events introduced

`generate_a3ss()`: A GRanges object with alternative 3' splice site events introduced

`generate_atss()`: A GRanges object with alternative transcription start site events introduced

`generate_ates()`: A GRanges object with alternative transcription end site events introduced

Examples

```
gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_se(gr, n_events = 1)


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_mxe(gr, n_events = 1)


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_ri(gr, n_events = 1)


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_a5ss(gr, n_events = 1)


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_a3ss(gr, n_events = 1)


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_atss(gr, n_events = 1)

# the non-overlapping variant: drop the first exon altogether
generate_atss(gr, n_events = 1, mode = "drop")


gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4
)
generate_ates(gr, n_events = 1)
```

---

get_seq *Extract sequences for GRanges*

Description

Given a GRanges, look up the DNA or RNA sequence of each range in a reference genome. This is a simple helper function that rearranges the `getSeq` arguments and combines it with other helper functions to refer to genomes by name.

Usage

```
get_seq(gr, genome, as_rna = FALSE)
```

Arguments

gr	A GRanges object.
genome	Either a character string naming an installed BSgenome package (e.g. "hg38", or "BSgenome.Hsapiens.UCSC.hg38") or a BSgenome object. See ?getBSgenome for details.
as_rna	If TRUE, return an RNAStringSet (T -> U). Default FALSE returns a DNAStringSet.

Value

A DNAStringSet (or RNAStringSet if as_rna = TRUE), one entry per range in gr, in the same order. Assign it onto gr (e.g. gr$seq <- ...) to keep it as a metadata column.

Examples

```
gr <- GenomicRanges::GRanges(
  "chr1", IRanges::IRanges(start = c(1e6, 1.1e6), width = 50)
)

suppressPackageStartupMessages(
  library(Biostrings)
)

get_seq(gr, "hg38")
get_seq(gr, "hg38", as_rna = TRUE)

set.seed(123)
gr <- create_mock_data(
  n_genes = 2, n_tx_per_gene = 3, n_exons_per_tx = 6
) |>
  generate_se(n_events = 1) |>
  GenomicRanges::shift(50e6) # move mock data

skipped <- find_se(gr)

suppressPackageStartupMessages(
  library(magrittr)
)
# magrittr pipe needed for the `.` placeholder below
skipped %>%
  plyranges::flank_upstream(100) %>%
  dplyr::mutate(seq = get_seq(., "hg38"))
```

---

prepare_exons *Prepare exon ranges from a TxDb and DTU results table*

Description

Extracts exon ranges from a TxDb object, merges them with differential transcript usage (DTU) results, and returns a flat GRanges ready for preprocess.

Usage

```
prepare_exons(
  txdb,
  dtu_table,
  coef_col,
  tx_id_col = "tx_id",
  gene_id_col = "gene_id",
  verbose = TRUE
)
```

Arguments

txdb	A TxDb object (from GenomicFeatures).
dtu_table	A data.frame or tibble with DTU results. Must contain columns for transcript ID, gene ID, and a coefficient.
coef_col	Column name in dtu_table with the coefficient / effect size values.
tx_id_col	Column name in dtu_table with transcript IDs matching the TxDb transcript names. Default "tx_id".
gene_id_col	Column name in dtu_table with gene IDs. Default "gene_id".
verbose	Whether to print progress messages. Default TRUE.

Value

A GRanges object with metadata columns: gene_id, tx_id, exon_rank, the coefficient column, and any additional columns from dtu_table.

Examples

```
library(AnnotationHub)
library(AnnotationDbi)
library(GenomicFeatures)
library(tibble)

ah <- AnnotationHub()
txdb <- ah[["AH84134"]] # fly TxDb (Drosophila melanogaster)

# build a simulated DTU table from the TxDb transcripts
txps <- txdb |>
  AnnotationDbi::select(
    keys(txdb, "TXID"), c("TXNAME", "GENEID"), "TXID"
  ) |>
  tibble::as_tibble() |>
  dplyr::select(tx_id = TXNAME, gene_id = GENEID)|>
  dplyr::filter(!is.na(gene_id))

sim_dtu_table <- txps |>
  dplyr::mutate(
    padj = runif(dplyr::n()),
    effect_est = rnorm(dplyr::n())
  )

fly_exons <- prepare_exons(
  txdb, sim_dtu_table, coef_col = "effect_est", verbose = TRUE
)
```

prepare_exons_by_partition

Prepare exons from two transcript partitions

Description

Combines two transcript partitions (up- and down-regulated) and assigns an `estimate` coefficient: `+1` to `up` and `-1` to `down`. Accepts either GRanges objects or character vectors of transcript IDs (in which case `txdb` is required to look up exon coordinates). The result is ready to pass to `preprocess` with `coef_col = "estimate"`.

Usage

```
prepare_exons_by_partition(
  up,
  down,
  txdb = NULL,
  tx_id_col = "TXNAME",
  verbose = TRUE
)
```

Arguments

`up`	A GRanges object or character vector of transcript IDs for the upregulated partition (assigned `estimate = +1`).
`down`	A GRanges object or character vector of transcript IDs for the downregulated partition (assigned `estimate = -1`).
`txdb`	A `TxDb` object (from GenomicFeatures). Required when `up` and `down` are character vectors of transcript IDs.
`tx_id_col`	The keytype in `txdb` matching the transcript IDs in `up` and `down`. Default `"TXNAME"`. Only used when `up` and `down` are character vectors. See `AnnotationDbi::keytypes(txdb` for available options.
`verbose`	Whether to print progress messages. Default `TRUE`.

Details

When `up` and `down` are GRanges, both must have `exon_rank`, `gene_id`, and `tx_id` metadata columns. Extra columns are kept; if one object lacks a column present in the other, those entries receive `NA`.

Value

A combined GRanges object with an `estimate` column (`+1` for `up`, `-1` for `down`), ready for `preprocess`.

Examples

```
# GRanges input
gr <- create_mock_data(n_genes = 1, n_tx_per_gene = 4, n_exons_per_tx = 4)
gr <- generate_se(gr, n_events = 1)
gr_down <- gr[gr$estimate < 0]
gr_up <- gr[gr$estimate > 0]
prepare_exons_by_partition(gr_up, gr_down) |>
```

```
  preprocess(coef_col = "estimate")
```

`preprocess` *Preprocess input GRanges object for splicing event calculation*

Description

This function checks that the input is a valid GRanges object with required metadata columns, then adds a unique key, the number of exons per transcript, and an 'internal' flag for each exon.

Usage

```
preprocess(gr, coef_col, method_string = NULL, additional_columns = NULL)
```

Arguments

`gr`	A GRanges object with metadata columns: 'exon_rank', 'gene_id', 'tx_id', 'coef'.
`coef_col`	The name of the metadata column indicating upregulated (+1) and downregulated (-1) exons.
`method_string`	The Differential Transcript Usage (DTU) method used to obtain the coef_col, for annotation purposes (optional).
`additional_columns`	A character vector of metadata column names to record for downstream use. Stored in `metadata(result)$additional_columns` (optional).

Value

A GRanges object with added 'key', 'nexons', and 'internal' columns.

Examples

```
# create mock data and run preprocessing
gr <- create_mock_data(n_genes = 2, n_tx_per_gene = 4, n_exons_per_tx = 4) |>
  preprocess(coef_col = "estimate", method_string = "mock_method")
```

Index

create_mock_data, 3

find_a3ss *(*find_events*)*, 4
find_a5ss *(*find_events*)*, 4
find_all_events *(*find_events*)*, 4
find_alternative_3_prime_splice_sites
        *(*find_events*)*, 4
find_alternative_5_prime_splice_sites
        *(*find_events*)*, 4
find_alternative_transcription_end_sites
        *(*find_events*)*, 4
find_alternative_transcription_start_sites
        *(*find_events*)*, 4
find_ates *(*find_events*)*, 4
find_atss *(*find_events*)*, 4
find_events, 4
find_ie *(*find_events*)*, 4
find_included_exons *(*find_events*)*, 4
find_mutually_exclusive_exons
        *(*find_events*)*, 4
find_mxe *(*find_events*)*, 4
find_retained_introns *(*find_events*)*, 4
find_ri *(*find_events*)*, 4
find_se *(*find_events*)*, 4
find_skipped_exons *(*find_events*)*, 4

generate_a3ss *(*generate_events*)*, 7
generate_a5ss *(*generate_events*)*, 7
generate_ates *(*generate_events*)*, 7
generate_atss *(*generate_events*)*, 7
generate_events, 7
generate_mxe *(*generate_events*)*, 7
generate_ri *(*generate_events*)*, 7
generate_se *(*generate_events*)*, 7
get_seq, 8

prepare_exons, 9
prepare_exons_by_partition, 11
preprocess, *9*, *11*, 12

splicelogic *(*splicelogic-package*)*, 2
splicelogic-package, 2