

# ASpli: An integrative R package for analysing alternative splicing using RNA-Seq

*Estefania Mancini, Andres Rabinovich, Javier Iserte, Marcelo Yanovsky, Ariel Chernomoretz*

April 9, 2020

## Contents

1	Introduction . . . . .	2
2	Getting started . . . . .	2
2.1	Installation . . . . .	2
2.2	Required data . . . . .	3
2.3	Building a TxDb of your genome . . . . .	3
3	Running ASpli. . . . .	3
3.1	Binning the genome . . . . .	4
3.1.1	Splicing event assignment. . . . .	4
3.1.2	binGenome method . . . . .	5
3.2	Defining the targets. . . . .	6
3.3	Read Counting . . . . .	7
3.3.1	Genomic alignment files loading . . . . .	7
3.3.2	readCounts: Counting feature overlapping reads . . . . .	8
3.3.3	Counting intron flanking regions . . . . .	8
3.3.4	Additional considerations . . . . .	9
3.3.5	The ASpliCounts object . . . . .	9
3.4	AsDiscover: A Junction-centered analysis of alternative splicing . . . . .	11
3.4.1	PIR and PSI metrics . . . . .	12
3.4.2	New splicing events discovery . . . . .	14
3.4.3	Contents of ASpliAS object . . . . .	14
3.5	Estimating differential gene expression and bin/junction usage . . . . .	15
3.5.1	Differential gene expression (DE) . . . . .	16
3.5.2	Differential bin and junction usage (DU) . . . . .	16
3.5.3	Usage of DUreport, DUreportBinSplice and junctionDUreport methods . . . . .	17
3.5.4	Contents of ASpliDU object . . . . .	21
3.6	Output and results . . . . .	23
3.7	Plots . . . . .	23
3.8	Utility functions . . . . .	26

3.8.1	Consulting data in ASpliDU objects . . . . .	26
3.8.2	Filtering ASpliCount objects . . . . .	28
3.8.3	Filtering ASpliDU objects . . . . .	28
3.8.4	Subsetting ASpli objects . . . . .	28
3.8.5	Merging DU and junction data for bins . . . . .	29
4	Example data . . . . .	29
4.0.1	Synthetic data set . . . . .	29
4.0.2	Real data set . . . . .	31

## 1 Introduction

---

Alternative splicing (AS) is a common mechanism of post-transcriptional gene regulation in eukaryotic organisms that expands the functional and regulatory diversity of a single gene by generating multiple mRNA isoforms that encode structurally and functionally distinct proteins. The development of novel high-throughput sequencing methods for RNA (RNA-Seq) has provided a powerful mean of studying AS under multiple conditions and in a genome-wide manner [1]. However, using RNA-Seq to study changes in AS under different experimental conditions is not trivial. In this vignette, we describe how to use ASpli, an integrative and user-friendly R package that facilitates the analysis of changes in both annotated and novel AS events. This package combines statistical information from exon, intron, and splice junction differential usage (p-value, FDR), with information from splice junction reads to calculate differences in the percentage of exon inclusion ( $\Delta$ PSI) and intron retention ( $\Delta$ PIR). The proposed methodology reliably reflect the magnitude of changes in the relative abundance of different annotated and novel AS events. This method can be used to analyze both simple and complex experimental designs involving multiple experimental conditions.

## 2 Getting started

---

### 2.1 Installation

ASpli is available at Bioconductor site and can be downloaded using `BiocManager::install()`:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
> BiocManager::install("ASpli")
```

```
> library(ASpli)
```

`BiocManager::install()` will take care of installing all the packages that ASpli depends on e.g. `edgeR`, `GenomicFeatures`, `GenomicRanges`, `GenomicAlignments`, `Gviz`, and other R package required. Some packages depend on operating system packages, like `curl`, that are not installed automatically, and should be installed by the user.

## 2.2 Required data

ASpli requires a single BAM file for each sample in the experiment. Also, a gene annotation file is mandatory (usually a GTF/GFF file). Make sure all files use the same coordinate system.

## 2.3 Building a TxDb of your genome

Gene annotation is handled using TxDb objects, which are provided by GenomicFeatures package. The building of a TxDb object might be time consuming, depending on the size of your genome. If a TxDb has been already created, we strongly recommend saving it as a sqlite file and reloading it any time the analysis is run. The GenomicFeatures package provides functions to create TxDb objects based on data downloaded from UCSC Genome Bioinformatics, BioMart or gff/gtf files. See "Creating New TxDb Objects or Packages" in the GenomicFeatures vignette for more details. In this example, a TxDb is built from a gtf file:

```
> library(GenomicFeatures)
> TxDb <- makeTxDbFromGFF(
  file="genes.gtf",
  format="gtf")
```

## 3 Running ASpli

The workflow intended to be used with ASpli is divided in five steps (Figure 1). For each of these steps ASpli provides a single or few functions to use:

1. Reading of genomic alignments: `loadBAM()`
2. Feature extraction: `binGenome()`
3. Read counting: `readCounts()`
4. Alternative splicing discovery using junctions: `AsDiscover()`
5. DE and DU estimation `DUreport()`, `junctionDUreport()` and `DUreportBinSplice()`

At each step it is possible to export results in a friendly manner. See Section 3.6:Output and results for more details.

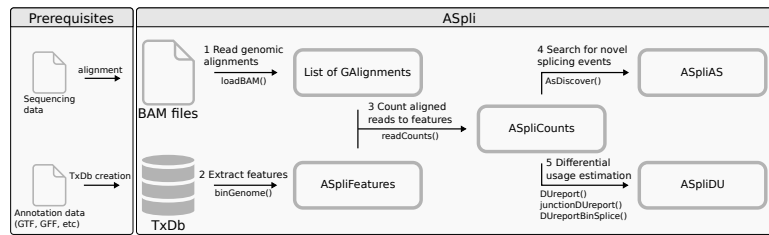
ASpli defines four major objects classes used to store data and perform operations related to a single task. Throughout this vignette each of these will be discussed in detail, however they are presented here:

**ASpliFeatures** Contains data extracted from annotation, such genomic coordinates of genes.

**ASpliCounts** Contains counts of aligned reads to each genomic feature.

**ASpliDU** Contains results of Differential expression and usage analysis.

**ASpliAS** Contains data useful to search novel splicing events.

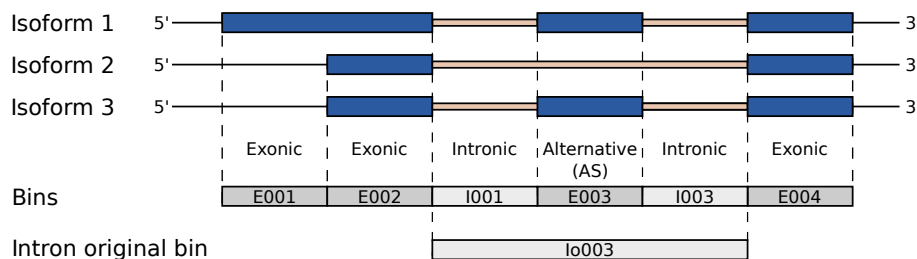


**Figure 1: Workflow of ASpli pipeline and prerequisites**

Each arrow represents one of the main steps of a typical analysis. Rounded boxes are objects created by ASpli functions. Names under arrows are functions included in ASpli used to generate each kind of object.

## 3.1 Binning the genome

Sub-genic features such as exons and introns are analyzed using existing annotations by splitting the genome into non-overlapping features called bins, following the methodology proposed for DEXSeq [2]. Exon and intron coordinates are then extracted from the annotation, but only those from multi-exonic genes are saved for further evaluation. When more than one isoform exists, some exons and introns will overlap. Exons and introns are then subdivided into new features called exon and intron bins, and are then classified into *exclusively exonic bins*, *exclusively intronic bins*, or *alternative splicing bins* (AS) (See Figure 2). In addition to these non overlapping bins, an additional type of bin is also defined. The latter corresponds to full introns extracted from annotation, before they are splitted by the presence of overlapping exons in another isoform. These bins as referred as *intron original bins* (Io).



**Figure 2: Schema of resulting bins from a gene with three hypothetical transcripts**

Those bins that are exonic and intronic in different isoforms are named *AS bins*.

### 3.1.1 Splicing event assignment

Each AS bin is assigned to a kind of splicing event. The assignment is made using *minimum local gene models* considering unique patterns of three bins: the bin being evaluated and the two neighbor bins (one to the left, one to the right). Figure (3) shows examples of splicing event categorization for different *minimum local gene models*.

When the minimum local gene model contains two isoforms (see first row of figure 3) bins are labelled as follow:

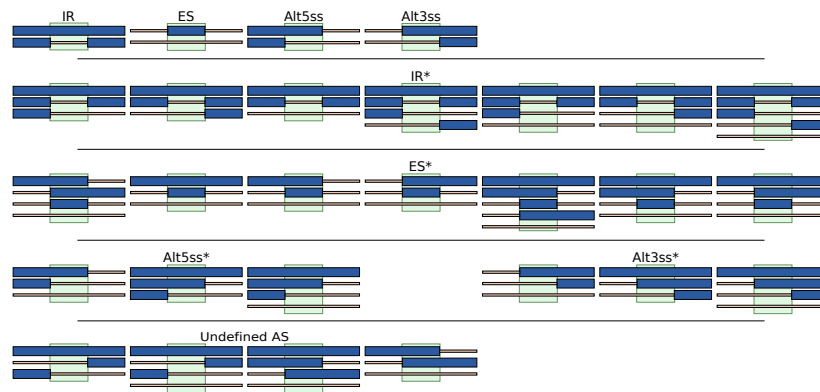
- **ES** Exon skipping
- **IR** Intron retention
- **Alt5'SS** Alternative 5' splicing site

- **Alt3'SS** Alternative 3' splicing site

When the minimum gene model contains three, four or five isoforms (no more than five non redundant isoforms can exist in the minimum gene model), labels of bins are similar to the ones defined previously but have an extra '\*' character (**IR\***, **ES\***, **Alt5'SS\***, **Alt3'SS\***) in order to show that more than two isoforms exists and, therefore, the results should be used with caution (see rows 2-4 of figure 3). This type of splicing events involve intronic subgenic regions surrounded by exons in at least one isoform (**IR\***), and exonic subgenic regions surrounded by two introns in at least one isoform (**ES\***). Minimum local gene models that do not fit in those categories are typified as **Alt5'SS\*** or **Alt3'SS\*** if they present exonic regions surrounded by intronic and exonic neighbor subgenic regions.

In some cases, it is not possible to make a clear assignation for a bin with assumptions made here, those bins are labelled as **Undefined AS** (last row of figure 3).

Additionally, those bins that overlap with the beginning or ending of any transcript are labelled as **external**. Please note that an external bin of a transcript may overlap to a non external bin of another transcript, in these cases the bin is still labelled as **external**.



**Figure 3: Summary of assignation of splicing events to bins from minimum gene model**

The bin being evaluated has a green background highlight. The blue boxes represents exons, while the little light orange boxes represent introns. Gene models shown are plus sense strand.

### 3.1.2 binGenome method

The `binGenome` method is a one-stop function to perform:

- subgenic splitting of genes into bins
- junction, gene and bin coordinates extraction
- bin splicing event assignation

Subgenic features are named as follow. For an hypothetical gene named GeneAAA:

- **GeneAAA:E001**: defines first exonic bin
- **GeneAAA:I001**: defines first intronic bin
- **GeneAAA:I001**: defines first `Intron original` before being divided
- **GeneAAA:J001**: defines first annotated junction of GeneAAA

## ASpli: An integrative R package for analysing alternative splicing using RNA-Seq

- **chr.start.end**: defines an experimental junction, **chr** is the chromosome name, **start** is the leftmost position of the junction and **end** is the rightmost position of the junction.

Bins and junctions are consecutively named from 5' to 3' sense of the plus strand of the reference sequence. This implies that bins and junctions with lower numbering are always closer to the 5' of that strand. Alternative splicing bins are named as exons.

`binGenome`'s output is an object of class `ASpliFeatures`.

Methods `featuresg`, `featuresb`, `featuresj` can be used to access genes, bins and junctions coordinates as `GRanges-List` (for the first case) or as `GRanges` objects (for bins and junctions). `GRangesList` and `GRanges` are objects defined in the `GenomicRanges` package. These objects contains additional information stored as metadata that can be accessed with `mcols` function.

```
> library(GenomicFeatures)
> annFile      <- aspliExampleGTF()
> aTxDb        <- makeTxDbFromGFF(annFile)
> # extract features from annotation
> features     <- binGenome( aTxDb )
> # Accesors of ASpliFeatures object
> geneCoord    <- featuresg( features )
> binCoord     <- featuresb( features )
> junctionCoord <- featuresj( features )
> # Extract metadata annotation, such as bins names, event and feature type
> binMetadata  <- mcols( binCoord )
```

```
> # extract features and map symbols to genes
> symbols      <- data.frame( row.names = genes( aTxDb ),
                             symbol = paste( 'This is symbol of gene:',
                                              genes( aTxDb ) ) )
> features     <- binGenome( aTxDb, geneSymbols = symbols )
> features
```

## 3.2 Defining the targets

Sample names, genomic alignment file names and experimental factors should be specified in a `data.frame` that has as many rows as samples. Each row should be named with the corresponding sample name. The first column must be named `bam` and should contain the path to a bam file. The remaining columns should be used to state the corresponding experimental factors for each sample. This `data.frame` is referred to as the **targets** `data.frame`.

For example, for a one factor experiment with two conditions (Control and Mutant) and two replicates for each condition, the **targets** `data.frame` could be defined as follow:

```
> bamFiles <- c( "Ct1.bam", "Ct2.bam", "Mut1.bam", "Mut2.bam" )
> targets <- data.frame( row.names = c("CT_rep1", "CT_rep2", "Mut_rep1", "Mut_rep2"),
                        bam = bamFiles,
                        genotype = c("CT", "CT", "Mut", "Mut") ,
                        stringsAsFactors = FALSE )

> targets

              bam genotype
CT_rep1    Ct1.bam      CT
```

CT_rep2	Ct2.bam	CT
Mut_rep1	Mut1.bam	Mut
Mut_rep2	Mut2.bam	Mut

More sophisticated designs should be specified as follow. For example: for a two factor experiment (factors genotype and time), with two values for each factor and two replicates for each condition, the **targets** `data.frame` can be defined as follow:

```
> bamFiles <- c("Ct_time1_rep1.bam", "Ct_time1_rep2.bam",
               "Ct_time2_rep1.bam", "Ct_time2_rep2.bam",
               "Mut_time1_rep1.bam", "Mut_time1_rep2.bam",
               "Mut_time2_rep1.bam", "Mut_time2_rep2.bam")
> targets_2 <- data.frame( row.names = c( 'CT_t1_r1', 'CT_t1_r2',
                                          'CT_t2_r1', 'CT_t2_r2',
                                          'Mut_t1_r1', 'Mut_t1_r2',
                                          'Mut_t2_r1', 'Mut_t2_r2' ),
                          bam = bamFiles,
                          genotype = c( 'CT', 'CT', 'CT', 'CT',
                                          'MUT', 'MUT', 'MUT', 'MUT' ),
                          time      = c( 't1', 't1', 't2', 't2',
                                          't1', 't1', 't2', 't2' ),
                          stringsAsFactors = FALSE )
> targets_2
```

	bam	genotype	time
CT_t1_r1	Ct_time1_rep1.bam	CT	t1
CT_t1_r2	Ct_time1_rep2.bam	CT	t1
CT_t2_r1	Ct_time2_rep1.bam	CT	t2
CT_t2_r2	Ct_time2_rep2.bam	CT	t2
Mut_t1_r1	Mut_time1_rep1.bam	MUT	t1
Mut_t1_r2	Mut_time1_rep2.bam	MUT	t1
Mut_t2_r1	Mut_time2_rep1.bam	MUT	t2
Mut_t2_r2	Mut_time2_rep2.bam	MUT	t2

The name of the each experimental condition is determined from the experimental factors defined in the **targets** `data.frame`. They can be consulted using the `getConditions` function.

```
> # Show the name of each unique condition in the console
> getConditions( targets_2 )
```

## 3.3 Read Counting

### 3.3.1 Genomic alignment files loading

Once the **targets** `data.frame` is defined the alignment data from bam files can be loaded easily with `loadBAM` function. BAM files are loaded and stored as a named list of `GAlignments` objects. `GAlignments` objects are defined in `GenomicAlignments` package.

```
> targets <- aspliTargetsExample()
> bam <- loadBAM(targets)
```

### 3.3.2 readCounts: Counting feature overlapping reads

The method `readCounts` allows to count the number of reads that overlaps each feature of interest (i.e. genes, bins, junctions, and intron/exon flanking regions). For genes and bins, read density values are also calculated considering the ratio between the number of overlapping reads and the length of a given feature. The following parameters can be specify:

- `features`: An `ASpliFeatures` object.
- `bam`: A list of `GAlignments` generated with `loadBAM` function.
- `targets`: A **targets** `data.frame` as defined in section 3.2 : [Defining the targets](#).
- `cores`: Number of processor cores used to speed up the computation.
- `readLength`: Read length of sequenced library
- `maxISize`: maximum intron expected size. Junctions longer than this size will be dicarded [3].
- `minAnchor`: minimum anchoring overlap for intron flanking regions (see 3.3.3 : [Counting intron flanking regions](#) ).

```
> counts <- readCounts (
  features,
  bam,
  targets,
  cores = 1,
  readLength = 100L,
  maxISize = 50000,
  minAnchor = 10 )
```

The result of `readCounts` method is an object of class `ASpliCounts`. Count and read density data could be extracted from it as `data.frame` objects using accesors methods. Also, it is possible to export count and read densities tables to text files in a tidy manner.

```
> # Accessing count and read density data
> GeneCounts <- countsg(counts)
> GeneRd <- rdsg(counts)
> BinCounts <- countsb(counts)
> BinRd <- rdsb(counts)
> JunctionCounts <- countsj(counts)
> # Export count data to text files
> writeCounts(counts=counts, output.dir = "example")
> writeRds(counts=counts, output.dir = "example")
```

### 3.3.3 Counting intron flanking regions

Every intron is considered as a potential retained intron. In order to analyze putative retention events for intron I, **ASpli** considers the corresponding adjacent 5' and 3' exons (E1 and E2, following the order in the genomic coordinates, regardless of the sense of the strand). Then, following [4], new artificial ranges that overlap the two retention regions E1I (connecting exon E1 and intron I) and IE2 (connecting intron I and exon E2) are defined:

- EI1: intron start - (readLength - minAnchor) — intron start + ( readLength - minAnchor )
- IE2: intron end - (readLength - minAnchor) — intron end + ( readLength - minAnchor )
- `minAnchor` is 10% of read length by default (parameter `minAnchor` )

Only those reads which minimum overlap `readLength` and without gap in this interval are considered. Accordingly, only sequences aligned within those two exons/introns are counted. If the reads of your genomic alignment are trimmed by quality rendering reads with a length inferior to the nominal read length, then `EI1` and `IE2` values will be underestimated.

```
> # Accessing counts for intron flanking regions
> eliCounts <- countseli(counts)
> ie2Counts <- countsie2(counts)
```

### 3.3.4 Additional considerations

For a given gene, the count number is computed as the number of reads overlapping any exon included in the corresponding annotated gene model. If a single read overlaps more than one exon, it is counted only once. Note that a read can overlap two different genes, in this case it is counted for both of them.

In order to estimate read densities, effective gene lengths are considered. The effective length is calculated as the sum of the length of exonic bins and alternative bins (i.e. all bins except intronic bins).

Junctions are extracted from BAM files. They are defined as those reads that aligned against disjoint region of the reference genome(N operator of CIGAR notation for aligned reads [5] ), and are essential piece of information for alternative splicing event quantification and discovery. Junction alignment quality/confidence is extremely important and it should be controlled at the moment of the alignment step.

### 3.3.5 The ASpliCounts object

Gene, bin and junction information stored in an `ASpliCounts` object can be accessed as stated in section (3.3.2) and (3.3.3). Acceser functions returns `data.frame` objects with the required information. They typically share the same general structure, but also include specific content depending on the nature of the requested feature. A more detailed overview is shown below:

- Gene counts (`countsg`) and gene read densities (`rdsg`) `data.frames` contain the following information (see table 1 for an example):

**row.names** Gene name as reported in annotation data.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section 3.1 : [Binning the genome](#)).

**locus\_overlap** Names of overlapping *loci*.

**gene\_coordinates** Genomic coordinates of the gene with format `chromosome:start-end`.

**start** Left-most position of the gene.

**end** Right-most position of the gene.

**length** Number of bases covering the gene.

**effective\_length** Number of bases that overlaps with annotated exons.

**sample data** Gene counts or read density (one column per sample).

- Bin counts (`countsb`) and bin read densities (`rdsb`) `data.frames` contain the following information:

**row.names** Bin name assigned according to the convention presented in [3.1.2 : binGenome method](#) .

**feature** Type of bin: **E** for exonic bins, **I** for intronic bins and **Io** for introns before splitting.

**event** Splicing event assigned to the bin (see section [3.1:Splicing event assignment](#))

**locus** The name of the locus that contains the bin.

**locus\_overlap** Names of all other overlapping *loci*.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section [3.1 : Binning the genome](#)).

**gene\_coordinates** Genomic coordinates of the gene with format `chromosome:start-end`.

**start** Left-most position of the bin.

**end** Right-most position of the bin.

**length** Number of bases covering the bin.

**sample data** Bin counts or read density (one column per sample) densities.

- Junction counts (`countsj`) `data.frame` contains the following information:

**row.names** Junction name in format `chromosome.start.end` (see [3.1.2 : binGenome method](#) ).

**junction** If junction coincides with a junction inferred from the annotation, the name is shown as is given in section [3.1.2:binGenome method](#) , otherwise it contains `noHit`.

**gene** Locus that contains the junction.

**strand** Sense strand of the gene.

**multipleHit** `yes` if junction spans multiple genes.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section [3.1 : Binning the genome](#)).

**gene\_coordinates** Genomic coordinates of the gene with format `chromosome:start-end`.

**bin\_spanned** Names of the bins spanned by the junction.

**j\_within\_bin** If junction falls within a single bin, the name of that bin is shown.

**sample data** Junction counts (one column per sample).

- E1l (`e1lcounts`) and IE2 (`ie2counts`) flanking region count `data.frame` contain the following information:

**row.names** Junction name in format `chromosome.start.end` (see [3.1.2 : binGenome method](#) ).

**junction** If junction coincides with a junction inferred from the annotation, the name is shown as is given in section [3.1:Binning the genome](#), otherwise contains `noHit`.

**gene** Name of the locus that contains the junction.

**strand** Strand sense of the gene.

**multipleHit** `yes` if junction spans multiple genes.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section [3.1:Binning the genome](#)).

**gene\_coordinates** Show the genomic coordinates of the gene with format `chromosome:start-end`.

**bin\_spanned** Names of the bins spanned by the junction.

**j\_within\_bin** If junction falls within a single bin, the name of that bin is shown.

**sample data** Junction counts (one column per sample).

Row names	symbol	locus_overlap	gene_coordinates	start	end	length	effective_length	Sample 1	Sample 2	...
GENE01	GENE01	-	chr1:1-700	1	700	700	700	324	314	n
GENE02	GENE02	-	chr1:1001-1800	1001	1800	800	550	327	333	n
GENE03	GENE03	-	chr1:2001-2800	2001	2800	800	650	342	321	n
GENE04	GENE04	-	chr1:3001-3800	3001	3800	800	650	313	337	n
...	...	...	...	...	...	...	...	...	...	...

Table 1: [Gene counts table example](#)

## 3.4 AsDiscover: A Junction-centered analysis of alternative splicing

Count data of bins and junctions can be used to get an integrative view of alternative splicing events. The method `AsDiscover` will produce an `ASpliAS` object containing comprehensive results that can be used as supporting evidence of alternative splicing usage of bins.

```
> as <- AsDiscover( counts, targets, features, bam, readLength=100L,
                    threshold = 5)
```

where:

- `counts`: An `ASpliCount` object.
- `targets`: A `targets` `data.frame` as defined in section 3.2 : [Defining the targets](#).
- `features`: An `ASpliFeatures` object (see 3.1.2 : [binGenome method](#) ).
- `bam`: A list of `GAlignments` generated with `loadBAM` function (see 3.3.1 : [Genomic alignment files loading](#)).
- `readLength`: Read length of sequenced library
- `threshold`: minimum number of reads supporting a junction.
- `cores`: Number of processor cores used to speed up the computation.

The analysis will consider junctions that are completely included within a unique gene and have more than a minimum number of reads supporting them, by default this number is five.

For each experimental junction identified, it is also reported if it is novel and which bins are spanned. In addition, it is stated if the junction is completely included in an annotated bin, which would indicate that the AS event is a possible *exintron* [6].

The contents of an `ASpliAS` object can be retrieved using accessor methods: `irPIR`, `altPSI`, `esPSI`, `joint junctionPIR`, `junctionPSI` (see section 3.4.3 : [Contents of ASpliAS object](#) for a detailed description of each method and retrieved data). In addition, method `writeAS`, allows to easily export all the data into text files tables.

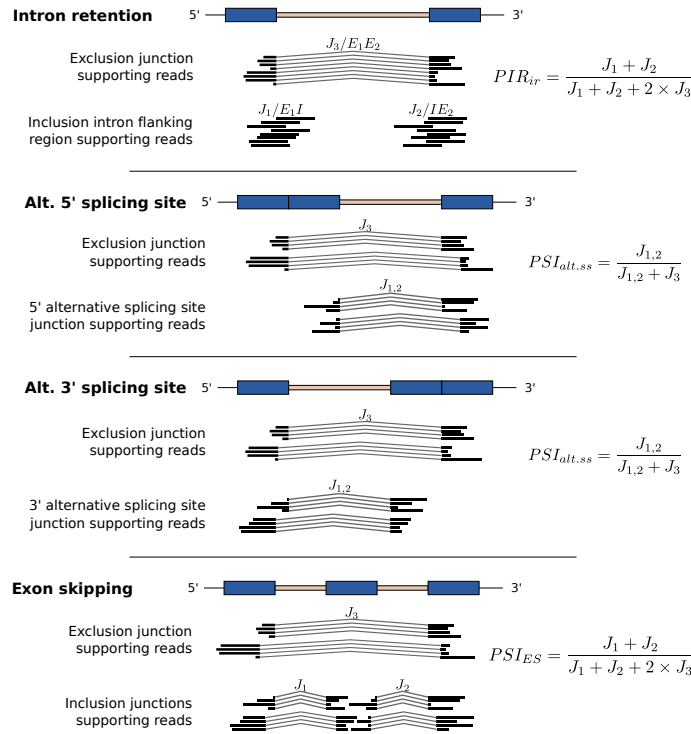
```
> # Access result from an ASpliAS object
> irPIR <- irPIR( as )
> altPSI <- altPSI( as )
> esPSI <- esPSI( as )
> junctionsPIR <- junctionsPIR( as )
> junctionsPSI <- junctionsPSI( as )
> allBins <- joint( as )
> # Export results to files
> writeAS(as=as, output.dir="example")
```

### 3.4.1 PIR and PSI metrics

To provide an integrative view of the AS events being analyzed, splice junction information is used to analyze differential bin usage. PSI (percent of inclusion) and PIR (percent of intron retention) metrics, which are extensively used to quantify AS [7], are calculated for bins junctions (see figure 4 ). Both, the PIR and PSI metrics, strongly enrich the count-centric analysis and provide independent experimental support for the existence of novel AS events, based on the identification of corresponding changes in nearby splice junctions.

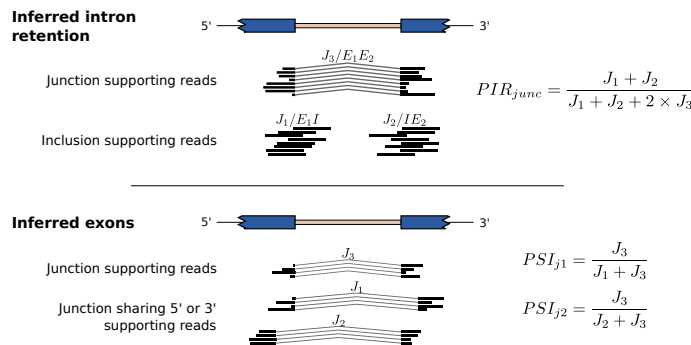
Only junctions that pass an abundance filter are considered for PSI/PIR calculation: a minimum number of counts (given by the `threshold` parameter) should be attained in all samples of at least one condition in order to consider the corresponding junction for further analysis (the default value of this threshold is five counts).

For each bin, a PIR or a PSI metric is calculated, depending on the splicing event category assigned to that bin (see section 3.1.1 : [Splicing event assignation](#)). If no splice event was assigned, meaning that the bin is not alternative, an exon will be considered to be involved in a putative exon skipping splicing event, and an intron will be considered to be involved in a putative intron retention splicing event.



**Figure 4: PSI and PIR metrics estimation and their relationship with junctions**

Every detected junction implies the existence of an intron that could be potentially retained. Therefore a PIR value is computed for every junction. Also, every junction defines a 5' and a 3' splicing site. Each one of these can be found exclusively in this junction or found also in other junctions. The latter is evidence of an exon skipping event or an alternative 5' or 3' splicing site event. Although it is not possible to infer the kind of event, the computation of a PSI value is helpful to detect this cases. The details of the formulas used to calculate these values are shown in figure 5.



**Figure 5: PSI and PIR metrics for junctions**

### 3.4.2 New splicing events discovery

ASpli allows novel AS events to be identified based on the splice junction repertoire. A novel AS event is identified whenever a novel splice junction that shares its beginning or end with another splice junction is discovered. When a novel AS event is identified using the splice junction repertoire, the PSI metric is calculated and reported. This ability to detect novel AS events and to estimate the magnitude of the changes in the usage of these AS events under different experimental conditions are original functions of the package.

### 3.4.3 Contents of ASpliAS object

Accessor methods of `ASpliAS` object return `data.frame` objects containing detailed information for bins involved in intron retention, 5' or 3' alternative splicing, exon skipping events, and for experimental junctions.

- Results for bins involved in intron retention, alternative 5' or 3' splicing, and exon skipping events can be accessed with `irPIR`, `altPSI` and `esPSI` respectively, or using `joint` method for all of them altogether. These methods return `data.frame` objects which report inclusion and/or exclusion evidence considering counts assigned to **J1**, **J2**, and **J3** type of junctions (see figures 4 and 5):

**row.names** Bin name as is defined in 3.1.2.

**event** Splicing event assigned to the bin (see section 3.1:Splicing event assignment)

**J1** The name of J1 junction or E/I flanking region.

**sample data for J1** J1 counts for each sample.

**J2** The name of J2 junction or E/I flanking region.

**sample data for J2** J2 counts for each sample.

**J3** The name of J3 junction or E/I flanking region.

**sample data for J3** J3 counts for each sample.

**PSI or PIR data** PSI variant metric or PIR metric (one column per sample).

- Results for experimental junctions assuming that they correspond to introns involved in intron retention events can be accessed with `junctionPIR` method. The returning value is a `data.frame` with columns:

**row.names** Junction name in format `chromosome.5'site-3'site`.

**hitIntron** If it exists, the name of a bin that exactly match the positions of the junction.

**hitIntronEvent** If `hitIntron` is not empty, the splicing event assigned to that bin.

**event** Splicing event assigned to the bin (see section 3.1:Splicing event assignment)

**sample data for J1** J1 counts for each sample.

**sample data for J2** J2 counts for each sample.

**sample data for J3** J3 counts for each sample.

**PIR data** PIR metric (one column per sample).

- Results for experimental junctions assuming that the bearing exons are involved in alternative splice events can be accessed with `junctionPSI` method. The returning value is a `data.frame` with columns:

**row.names** Junction name in format `chromosome.5'site-3'site`.

**junction** If junction coincides with a junction inferred from the annotation, the name is shown as is given in section 3.1: [Binning the genome](#), otherwise contains `noHit`.

**gene** The name of the locus that contains the junction.

**strand** Sense strand of the gene.

**multipleHit** `yes` if junction spans multiple genes.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section 3.1 : [Binning the genome](#)).

**gene\_coordinates** Show the genomic coordinates of the gene with format `chromosome:start-end`.

**bin\_spanned** The names of the bins spanned by the junction.

**j\_within\_bin** If junction fall within a single bin, the name of that bin is shown.

**sample data for J3** J3 counts for each sample.

**StartHit** The name of J1 junction.

**sample data for J1** J1 counts for each sample.

**PSI data** A column by condition with the PSI metric for J1. The name of the column is the name of the condition with 'start' suffix.

**EndHit** The name of J2 junction.

**sample data for J2** J2 counts for each sample.

**PSI data** A column by condition with the PSI metric for J2. The name of the column is the name of the condition with 'end' suffix.

## 3.5 Estimating differential gene expression and bin/junction usage

ASpli provides two methodologies (`DUreport` and `DUreportBinSplice`) to perform statistical analysis of differential gene expression and differential usage of bins. Both functions implements the same gene-level analysis. However, while `DUreportBinSplice` is a wrapper to the `diffSpliceDGE` function from edgeR, `DUreport` implements a pseudocount normalization method that provides similar detection performance. ASpli also provides a third function, `junctionDUreport`, that only takes into account junction information.

Results are stored in `ASpliDU` objects and can be accessed for bins, junctions, and genes using the accessors methods `binsDU`, `junctionDU`, and `genesDE` respectively. Results can also be exported as text files tables using the `wroteDU` function.

### 3.5.1 Differential gene expression (DE)

The first step of the gene DE analysis pipeline is the selection of those annotated genes that are considered to be expressed, or at least that can be confidently detected. For this, the average number of reads for each gene and condition and the average read density for each gene and condition are calculated as shown below:

$$\hat{G}_{ij}^c = \sum_{k \in K_j} \frac{S_{ik}^c}{|K_j|}$$

$$\hat{G}_{ij}^{rd} = \sum_{k \in K_j} \frac{S_{ik}^{rd}}{|K_j|}$$

Where  $\hat{G}_{ij}^c$  is avg. gene count for gene  $i$  in condition  $j$ ,  $\hat{G}_{ij}^{rd}$  is avg. read density for gene  $i$  in condition  $j$ ,  $S_{ik}^c$  is gene count for gene  $i$  in sample  $k$ ,  $S_{ik}^{rd}$  is read density for gene  $i$  in sample  $k$ .  $K_j$  is the set of samples corresponding to condition  $j$ .

Gene  $i$  is considered expressed if any of the corresponding  $\hat{G}_{ij}^c$  and  $\hat{G}_{ij}^{rd}$  values are equal or greater than a minimum cutoff (by default, these numbers are 10 for gene count and 0.05 for read density) and for any condition  $j$ .

Differential gene expression is estimated with edgeR [8] package using an exact test (for paired comparison) or generalized linear model (GLM) for more complex designs. For each gene **logFC**, **pvalue**, and **adjusted pvalue** by false discovery rate [9] are reported.

DUreport and DUreportBinSplice methods perform genes DE analysis with the same methodology and gives the same results.

Each condition in the analysis is weighted by contrast vector that ultimately defines the comparison that will be done. For a simple pair comparison between `Control` and `Trat` conditions, represented as  $Trat - Control$  can be expressed as:

```
> contrast <- c( -1, 1 )
```

Assuming that the order of the conditions given by `getConditions` methods is `[ 'Control', 'Trat' ]`.

### 3.5.2 Differential bin and junction usage (DU)

Bins and junctions are treated similarly for DU analysis. In order to select informative events, only bins and junctions from expressed genes (see section 3.5.1 : [Differential gene expression \(DE\)](#)) will be taken into account if they have more reads than a threshold (five reads by default) supporting them in at least one of the conditions.

External bins do not participate in alternative splicing, therefore are usually excluded from the analysis. However, an external bin in one isoform can overlap to a non external bin in other isoform that can participate in alternative splicing regulation. ASpli allows to optionally include the external bins in the analysis. Also, original intron bins can be optionally excluded. The inclusion of those bins can affect the estimation of corrected p-values (fdr). The information provided by intron original bins are highly correlated with the one of their sub bins, and increase largely the number of events to be analyzed. Therefore the fdr correction is more

strict. Also, there is a violation on the *fdr* correction assumption that all individual tests are independent from each other. If a intron original bin show a significant change, there is a very high chance that at least one of their sub bins also shows a significant change.

Differential bin/junction usage are estimated using the statistical methodology proposed in the `edgeR` package. Read counts from bins/junctions can not be used naively for differential usage analysis. Counts needs to be adjusted to remove the signal from changes in gene expression. To perform this, `ASpli` provides three alternatives:

1. Adjust bin/junction counts to gene counts
2. Use an offset value given by gene expression
3. Delegate differential bin usage estimation to `edgeR`'s `diffSpliceDGE`. This is available only for bins.

The adjustment of bin counts to gene counts is performed by transforming the raw bin counts. The transformation is done by:

$$B_{ijk}^A = \frac{B_{ijk} \times \bar{G}_j}{G_{jk}} \quad \mathbf{1}$$

Where  $B_{ijk}^A$  is the adjusted bin count for bin  $i$  of gene  $j$  in sample  $k$ ,  $B_{ijk}$  is the raw bin count for bin  $i$  of gene  $j$  in sample  $k$ ,  $\bar{G}_j$  is the mean raw count for gene  $j$  for all samples and  $G_{jk}$  is the raw gene count for gene  $j$  in sample  $k$ .

Then the adjusted bin counts are used as input for an statical analysis identical to the one performed on genes using the methods provided by `edgeR` package.

The offset methodology consist in define a vector that reflects the change of gene expression levels and use it to correct the observed change in bin usage in the statistical analysis. There are several ways to estimate the offset vector:

- Use gene counts (gene mode):
  1. Use only raw counts.
  2. Fit a generalized linear model from gen counts.
- Use the sum of exonic bins from each gene (bin mode). Is important to note that in this mode some reads are counted more than one time, because the overlap more than one bin. This mode is not available for junctions.

`EdgeR` package also provides a separate framework to detect splicing events, this rely on their `diffSpliceDGE` method. This method uses an strategy similar to the offset methodology. `ASpli` allow its use with the method `DUreportBinSplice`.

### 3.5.3 Usage of `DUreport`, `DUreportBinSplice` and `junctionDUreport` methods

`DUreport` method performs DE estimation for genes and DU estimation for bins in the same run. The call of this method is done by:

```
> du <- DUreport( counts,
                  targets,
                  minGenReads = 10,
                  minBinReads = 5,
```

```
minRds = 0.05,  
offset = FALSE,  
offsetAggregateMode = c( "geneMode", "binMode" )[2],  
offsetUseFitGeneX = TRUE,  
contrast = NULL,  
forceGLM = FALSE,  
ignoreExternal = TRUE,  
ignoreIo = FALSE,  
ignoreI = FALSE,  
filterWithConstrasted = FALSE,  
verbose = FALSE )
```

Where the accepted parameters are:

**counts** an ASpliCounts object.

**targets** a targets data frame built as explained in [3.2 : Defining the targets](#).

**minGenReads** minimum value for reads in any condition required for a gene to be considered as expressed (see [3.5.1 : Differential gene expression \(DE\)](#) ). The default value is 10.

**minBinReads** minimum value for reads in any condition required for a bin to be included in the analysis (see [3.5.2 : Differential bin and junction usage \(DU\)](#) ). The default value is 5.

**minRds** minimum value for read density for genes and bins required to be considered in the analysis. The default value is 0.05.

**offset** a logical value that indicates, if **TRUE**, that an offset vector should be estimated and used in the analysis. If **FALSE**, bin counts by gene counts adjustment is used instead. The default value is **FALSE** ( see [3.5.2 : Differential bin and junction usage \(DU\)](#) )

**offsetAggregateMode** a character value that specifies if gene mode or bin mode of offset estimation should be used. It is only relevant when

**offset** argument is **TRUE** (see [3.5.2 : Differential bin and junction usage \(DU\)](#)). The default value is **geneMode**.

**offsetUseFitGeneX** a logical value that indicates if the offset vector should be computed using a GLM. This is only relevant when **offset** argument is **TRUE** and **offsetAggregateMode** argument is **geneMode** (see [3.5.2 : Differential bin and junction usage \(DU\)](#)). The default value is **TRUE**.

**contrast** is a vector representing the coefficients for each unique condition in the analysis. They are assigned in the order given by **getConditions** method (see [3.2 : Defining the targets](#)). The default value is **NULL** which correspond to a vector with value -1 in the first position, 1 in the second, and zero for the remaining positions. This is for the paired comparison of the second condition versus the first condition.

**forceGLM** a logical value that indicates that fitting a GLM should be used in the analysis for paired comparison, instead of an exact test (used by default for this type of comparison). The default value is **FALSE**.

**ignoreExternal** a logical value indicating that external bins should be ignored in the analysis. The default value is **TRUE**.

**ignoreIo** a logical value indicating that original intron bins should be ignored in the analysis. The default value is **TRUE**.

**ignoreI** a logical value indicating that only exons should be included in the analysis. The default value is `FALSE`.

**filterWithConstrasted** a logical value that specifies that gene and bin filtering when more than two conditions are present, should be done only with the data of those conditions and ignore the others. The default value is `FALSE`, it is strongly recommended to do not change this value.

**verbose** a logical value that indicates that detailed information about each step in the analysis will be presented to the user.

`DUreportBinSplice` method performs DE estimation for genes and DU estimation for bins in the same run. The call of this method is done by:

```
> du <- DUreportBinSplice( counts,
                           targets,
                           minGenReads = 10,
                           minBinReads = 5,
                           minRds = 0.05,
                           contrast = NULL,
                           forceGLM = FALSE,
                           ignoreExternal = TRUE,
                           ignoreIo = TRUE,
                           ignoreI = FALSE,
                           filterWithConstrasted = FALSE )
```

Where the accepted parameters are:

**counts** an `ASpliCounts` object.

**targets** a targets data frame built as explained in [3.2 : Defining the targets](#).

**minGenReads** minimum value for reads in any condition required for a gene to be considered as expressed (see [3.5.1 : Differential gene expression \(DE\)](#) ). The default value is 10.

**minBinReads** minimum value for reads in any condition required for a bin to be included in the analysis (see [3.5.2 : Differential bin and junction usage \(DU\)](#) ). The default value is 5.

**minRds** minimum value for read density for genes and bins required to be considered in the analysis. The default value is 0.05.

**contrast** is a vector representing the coefficients for each unique condition in the analysis. They are assigned in the order given by `getConditions` method (see [3.2 : Defining the targets](#)). The default value is `NULL` which correspond to a vector with value -1 in the first position, 1 in the second, and zero for the remaining positions. This is for the paired comparison of the second condition versus the first condition.

**forceGLM** a logical value that indicates that fitting a GLM should be used in the analysis for paired comparison, instead of an exact test (used by default for this type of comparison). The default value is `FALSE`.

**ignoreExternal** a logical value indicating that external bins should be ignored in the analysis. The default value is `TRUE`.

**ignoreIo** a logical value indicating that original intron bins should be ignored in the analysis. The default value is `TRUE`.

**ignoreI** a logical value indicating that only exons should be included in the analysis. The default value is `FALSE`.

**filterWithContrasted** a logical value that specifies that gene and bin filtering when more than two conditions are present, should be done only with the data of those conditions and ignore the others. The default value is `FALSE`, it is strongly recommended to do not change this value.

`junctionDUreport` method performs DU estimation for junctions. This is independent of the genes DE and bins DU analysis. However, the results can be merged with existing genes and bins results. The call of this method is done by:

```
> du <- junctionDUreport( counts,
                          targets,
                          appendTo = NULL,
                          minGenReads = 10,
                          minRds = 0.05,
                          threshold = 5,
                          offset = FALSE,
                          offsetUseFitGeneX = TRUE,
                          contrast = NULL,
                          forceGLM = FALSE )
```

Where the accepted parameters are:

**counts** an `ASpliCounts` object.

**targets** a targets data frame built as explained in [3.2 : Defining the targets](#).

**appendTo** an `ASpliDU` object to which the results will be attached.

**minGenReads** minimum value for reads in any condition required for a gene to be considered as expressed (see [3.5.1 : Differential gene expression \(DE\)](#) ). The default value is 10.

**minRds** minimum value for read density for genes required to be considered as expressed. The default value is 0.05.

**threshold** minimum number of junction supporting reads required to be present in at least one sample of every conditions in order to include the junction in the analysis.

**offset** a logical value that indicates, if `TRUE`, that an offset vector should be estimated and used in the analysis. If `FALSE`, junction counts by gene counts adjustment is used instead. The default value is `FALSE` ( see [3.5.2 : Differential bin and junction usage \(DU\)](#) )

**offsetUseFitGeneX** a logical value that indicates if the offset vector should be computed using a GLM. This is only relevant when `offset` argument is `TRUE` (see [3.5.2 : Differential bin and junction usage \(DU\)](#)). The default value is `TRUE`.

**contrast** is a vector representing the coefficients for each unique condition in the analysis. They are assigned in the order given by `getConditions` method (see [3.2 : Defining the targets](#)). The default value is `NULL` which correspond to a vector with value -1 in the first position, 1 in the second, and zero for the remaining positions. This is for the paired comparison of the second condition versus the first condition.

**forceGLM** a logical value that indicates that fitting a GLM should be used in the analysis for paired comparison, instead of an exact test (used by default for this type of comparison). The default value is `FALSE`.

The three methods described above return an `ASpliDU` object. This object contains the results for genes DE, bins DU and junction DU, which can be consulted through their respective accessor methods: `genesDE`, `binsDU` and `junctionsDU`. See next section (3.5.4 : [Contents of ASpliDU object](#)) for details about the specific contents of the data obtained with these methods.

### 3.5.4 Contents of ASpliDU object

ASpliDU accessor methods `genesDE`, `binsDU` and `junctionsDU` returns a `data.frame` object with the corresponding results.

- Results for genes differential expression analysis can be accessed with `genesDE`. This method returns a `data.frame` object which columns are specified below.

**row.names** Gene name as is found in annotation.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section 3.1 : [Binning the genome](#)).

**locus\_overlap** Show the names of all other overlapping *loci*.

**gene\_coordinates** Show the genomic coordinates of the gene with format `chromosome:start-end`.

**start** Left-most position of the gene.

**end** Right-most position of the gene.

**length** Number of bases covering the gene.

**effective\_length** Number of bases that overlaps with annotated exons.

**logFC** Show the observed change of expression level in a  $\log_2$  scale.

**pvalue** The pvalue for the DE test.

**gen.fdr** The pvalue adjusted by **Benjamini-Hochberg** method.

- Results for experimental junctions assuming that they correspond to introns involved in intron retention events can be accessed with `junctionPIR` method. The returning value is a `data.frame` with columns:

**row.names** Bin name as given in section 3.1.2: [binGenome method](#) .

**feature** Is the kind of bin: **E** for exonic bins, **I** for intronic bins and **lo** for introns before splitting.

**event** Splicing event assigned to the bin (see section 3.1: [Splicing event assignment](#))

**locus** The name of the locus that contains the bin.

**locus\_overlap** Show the names of all other overlapping *loci*.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section 3.1: [Binning the genome](#)).

**gene\_coordinates** Show the genomic coordinates of the gene with format `chromosome:start-end`.

**start** Left-most position of the bin.

**end** Right-most position of the bin.

**length** Number of bases covering the bin.

**logFC** Show the observed change of usage in a  $\log_2$  scale.

**pvalue** The pvalue for the DU test.

**gen.fdr** The pvalue adjusted by **Benjamini-Hochberg** method.

- Results for experimental junctions assuming that the bearing exons are involved in alternative splice events can be accessed with `junctionPSI` method. The returning value is a `data.frame` with columns:

**row.names** Junction name in format `chromosome.5'site-3'site`.

**junction** If junction coincides with a junction inferred from the annotation, the name is shown as is given in section [3.1.2:binGenome method](#), otherwise contains `noHit`.

**gene** The name of the locus that contains the junction.

**strand** Sense strand of the gene.

**multipleHit** Contains `yes` if junction spans multiple genes.

**symbol** An optional common name for the gene, that must be provided at the moment of feature extraction (see section [3.1:Binning the genome](#)).

**gene\_coordinates** Show the genomic coordinates of the gene with format `chromosome:start-end`.

**bin\_spanned** The names of the bins spanned by the junction.

**j\_within\_bin** If junction fall within a single bin, the name of that bin is shown.

**junction count data** One column per sample containing raw junction counts if `offset` argument is `TRUE`, or junction counts adjusted by gene counts otherwise.

**logFC** Show the observed change of usage in a  $\log_2$  scale.

**pvalue** The pvalue for the DU test.

**gen.fdr** The pvalue adjusted by **Benjamini-Hochberg** method.

**bin\_start\_hit** List of junction names that shares the start position with current junction, in format `chromosome:start-end`.

**junction sharing start data** One column per samples containing the counts of junctions sharing the start with the current junction. Junction counts can be raw or adjusted following the same criteria described for `junction count data` column.

**junction sharing start ratio** One column by condition containing the ratio of current junction among all junctions sharing their start position (see [3.4.1 : PIR and PSI metrics](#)). The ratio is calculated using raw or adjusted counts.

**end\_start\_hit** List of junction names that shares the start position with current junction, in format `chromosome:start-end`.

**junction sharing end data** One column per samples containing the counts of junctions sharing the end with the current junction. Junction counts can be raw or adjusted following the same criteria described for `junction count data` column.

**junction sharing end ratio** One column by condition containing the ratio of current junction among all junctions sharing their end position (see [3.4.1 : PIR and PSI metrics](#)). The ratio is calculated using raw or adjusted counts.

## 3.6 Output and results

At each module, results are stored in `ASpli0bjects`. Self-explanatory tables can be exported at each step of the analysis. Using `write` functions, it is possible to export tab delimited tables in a features-level output folder:

```
> writeCounts(counts, "example_counts")
> writeDU(du, output.dir="example_du");
> writeAS(as=as, output.dir="example_as");
> writeAll(counts=counts, du=du, as=as, output.dir="example_all")
```

Column of text tables has the same name and meaning that those described in [3.3.5 : The ASpliCounts object](#), [3.4.3 : Contents of ASpliAS object](#) and [3.5.4 : Contents of ASpliDU object](#). Exported text tables files are arranged in subfolders "exons", "genes", "introns" and "junctions". `writeAS` creates an additional text table:

**as\_discovery.tab** Contains junction counts and PSI/PIR metrics for all bins.

`writeAll` methods writes some additional tables:

**summary.tab** Contains bin DU usage for all bins together.

**bins\_du\_psi\_pir.tab** Contains bin DU usage for all bins merged with PSI/PIR metrics.

## 3.7 Plots

Inspection of coverage plots is a good practice for the interpretation of analytical results. After selection of AS candidates it is possible to plot the results in a genome browser manner highlighting alternatively spliced regions using the function `plotGenomicRegions`. Coverage data is taken from **bam** files. `plotGenomicRegions` function will draw a coverage plot for each condition, not for each sample, therefore coverage data from samples needs to be merged. `plotGenomicRegions` can extract and merge the reads of the corresponding *locus* of interest on-the-fly from sample's **bam** files or can use previously merged **bam** generated by the user. When the number of *loci* to be plotted is small (ten or less) the first approach is usually faster, otherwise is better to merge and index **bam** files beforehand. **Bam** files can be merged using `samtools` utility or `RSamtools` package.

```
> library(GenomicFeatures)
> bamfiles <- system.file( 'extdata',
  c('A_C_0.bam', 'A_C_1.bam', 'A_C_2.bam',
    'A_D_0.bam', 'A_D_1.bam', 'A_D_2.bam',
    'B_C_0.bam', 'B_C_1.bam', 'B_C_2.bam',
    'B_D_0.bam', 'B_D_1.bam', 'B_D_2.bam' ),
  package = "ASpli")
> targets <- data.frame(
  row.names = c( 'A_C_0', 'A_C_1', 'A_C_2',
```

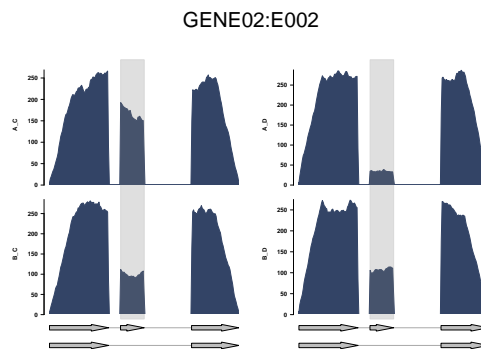
```

      'A_D_0', 'A_D_1', 'A_D_2',
      'B_C_0', 'B_C_1', 'B_C_2',
      'B_D_0', 'B_D_1', 'B_D_2' ),
bam = bamfiles,
f1 = rep( c('A','B'), each=6 ),
f2 = rep( c('C','D'), 2, each=3 ),
stringsAsFactors = FALSE )
> genomeTxDb <- makeTxDbFromGFF( system.file( 'extdata', 'genes.mini.gtf',
package='ASpli' ))
> features <- binGenome( genomeTxDb )
> # Draw a single plot on a window
> # xIsBin arguments is used to specify if names given are bins or genes.
> # When multiple names are given, all must correspond to bins or correspond to
> # genes. Mixed names are not allowed.
>
> plotGenomicRegions( features, 'GENE02:E002' , genomeTxDb, targets,
  xIsBin = TRUE, verbose = TRUE)
> # Draw a single plot on a window, with a custom layout
> plotGenomicRegions( features, 'GENE02:E002' , genomeTxDb, targets, xIsBin =
  TRUE, layout = matrix( c('A_C','A_D','B_C','B_D') ,nrow = 1 ),
  verbose = TRUE, color = '#AA8866')
> # Draw many plots on files.
> # Argument outfileType define the file format of the generated files.
> # Argument useTransparency specified that transparency can be used to draw the
> # plot. No all graphic devices support transparency.
> # Argument annotationHeight specifies the proportional height of the plot used
> # to draw the annotation.
> # Argument deviceOpt gives additional arguments (width and height in this
> # example) to the underlying graphic device (pdf in this example)
> # One of the plots generated here is shown in a figure below.
> plotGenomicRegions( features, c( 'GENE02:E002', 'GENE01:E002', 'GENE02:E003' ),
  genomeTxDb, targets, xIsBin = TRUE, outfolder = 'grplots', verbose = TRUE,
  color = '#334466', outfileType='pdf', useTransparency = TRUE,
  annotationHeight = 0.12, deviceOpt = c( width = 8, height = 6 ) )
> # Draw a single plot on a window, with premerged bams
> mergedBams <- data.frame(
  row.names = c( 'A_C','A_D','B_C','B_D' ),
  bam = c( 'A_C.bam', # Warning, these files do not exists.
    'A_D.bam', # Bam files should be indexed.
    'B_C.bam',
    'B_D.bam' ))
> plotGenomicRegions( features, 'GENE02:E002' , genomeTxDb, targets,
  xIsBin = TRUE, verbose = TRUE, preMergedBAMs = mergedBams )

```

Figure 6 shows a plot from one of the previous examples.

In addition to coverage plots, ASpli is able to draw plots containing raw gene counts, raw bin counts, PSI/PIR value, inclusion and exclusion junction counts for a single bin in all, or a subset, conditions of the analysis. This plot is useful to integrate evidences of different sources in order to explain an experimental result. The function to make these plots is `plotBins`, below there are some examples of use.



**Figure 6:** Genomic plot of a gene containing an exon skipping splicing event highlighted for four different conditions

Example 1:

```
> # Defines an experiment with one experimental factor (genotype, with two
> # values: wild-type (WT) and mutant (MT) ), and three replicate samples for each
> # condition.
> targets <- data.frame(
  row.names = c( 'WT1', 'WT2', 'WT3',
                 'MT1', 'MT2', 'MT3' ),
  bam = c( 'WT1.bam', 'WT2.bam', 'WT3.bam',
            'MT1.bam', 'MT2.bam', 'MT3.bam' ),
  genotype = c( 'WT', 'WT', 'WT', 'MT', 'MT', 'MT' ),
  stringsAsFactors = FALSE )
> # Specifies what factors and values will be plotted, in this example all of
> # them.
> fv = list( genotype = c( 'WT', 'MT' ) )
> plotbins(
  counts,
  as,
  'GENE02:E002',
  factorsAndValues = fv,
  targets )
```

Example 2:

```
> # Defines an experiment with two experimental factor (treat, with four
> # values: A, B, C and D; and time, with twelve values from T1 to T12.), and
> # two replicate samples for each condition. In the definition, many values are
> # not shown to reduce the space of the example, and are replaced by '...' .
> targets <- data.frame(
  row.names = c( 'A_T1.r1', 'A_T1.r2',
                 'A_T2.r1', 'A_T2.r2',
                 ...,
                 'D_T12.r1', 'D_T12.r2' ),
  bam = c( 'A_T1.r1.bam', 'A_T1.r2.bam',
            'A_T2.r1.bam', 'A_T2.r2.bam',
            'D_T12.r1.bam', 'D_T12.r2.bam' ) ,
```

```

    treat = c( 'A', 'A',
               'A', 'A',
               ... ,
               'D', 'D' ),
    time = c( 'T1', 'T1',
              'T2', 'T2',
              ... ,
              'T12', 'T12' ),
    stringsAsFactors = FALSE )
> # Draw the plots.
> plotbins(
    counts,
    as,
    'GENE02:E002',
    factorsAndValues = fv,
    targets )
> # Specifies what factors and values will be plotted. In this example there are
> # two factor, the first one in 'fv' list is the main factor, values will
> # be grouped by this factor.
> fv = list( time = c( 'T1', 'T2', ... , 'T12' ) ,
             treat = c( 'A', 'B', 'C', 'D' ) )
> # Draw the plots
> plotbins(
    counts,
    as,
    'GENE02:E002',
    factorsAndValues = fv,
    targets )

```

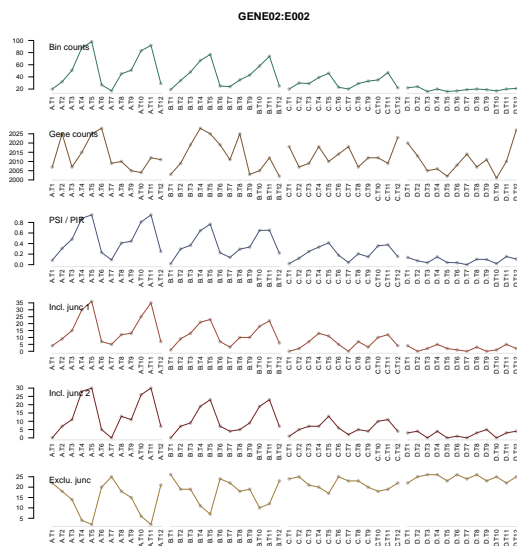
Figures 7 and 8 shows the images that can be obtained in setups like shown in the examples above. Note that one image contains barplots, and the other contains lines. They are selected automatically according to the number of conditions and values per condition. However, it is possible to force to one of them setting the argument `useBarPlots` in `plotBins` method.

## 3.8 Utility functions

ASpli includes many functions that help the use to manipulate and interrogate the different ASpli objects.

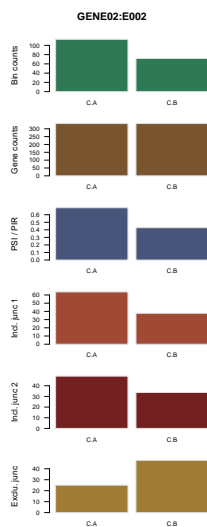
### 3.8.1 Consulting data in ASpliDU objects

ASpliDU objects can contain data for gene, bins and/or junctions depending on what function was used to create it. `DUreport` and `DUreportBinSplice` methods returns `ASpliDU` objects with data for gene and bins. `junctionDUreport` function returns an object with junction data only or with gene, bins and junction depending on the input. There are two function that are useful to know what kind of data an `ASpliDU` contains: `containsJunctions` and `containsGenesAndBins`.



**Figure 7: Plots of bin gene, and junction counts, and PSI/PIR metric for a single bin of an experiment containing a time course of 12 posints in four conditions**

Gene expression seem to be constant across all conditions, however, the usage of the bin oscilates with time. Inclusion and exclusion junctions show a concordant behaviour.



**Figure 8: Plots of bin gene, and junction counts, and PSI/PIR metric for a single bin of an experiment containing two combinatorial factors**

Of those, only two conditions are shown.

```
> du <- aspliDUexample1()
> containsJunctions( du )
> containsGenesAndBins( du )
```

### 3.8.2 Filtering ASpliCount objects

Genes, bins and junctions present in an `ASpliCount` object can be filtered by abundance according to several statements:

- number of counts or read density.
- mean value or minimum value.
- grouping by condition or on the whole set.
- all conditions must meet the selection criteria or at least one condition must do it.

`filterReadCounts` method allow this filtering. Here there are some examples:

### 3.8.3 Filtering ASpliDU objects

Results present in `ASpliDU` can be filtered by **logFC** and **FDR** with `filterDU` function. Elements with and **FDR** value lower or equal to a given threshold, and with an absolute **logFC** greater or equal to a given value are kept. Default values for **logFC** and **FDR** threshold values are 0 and 1 respectively, in other words, is not filtered by that value.

```
> # Filter genes that changes at least a 50 percent their expression with a
> # FDR value of 0.05 or less, and bins changes at least a 50 percent their expression with a
> # FDR value of 0.05
> du <- aspliDUexample1()
> duFilt <- filterDU(du, 'genes', '0.05', log(1.5,2))
> duFilt <- filterDU(duFilt, 'bins', '0.1', log(1.5,2))
```

It is possible to filter only those elements that has a positive (or negative) **logFC** using the arguments `absLogFC` and `logFCgreater`. `absLogFC` forces to take the absolute value of **logFC**, this is `TRUE` by default. `logFCgreater` specifies, if `TRUE`, that the **logFC** value must be greater than the threshold.

```
> # Filter genes that reduces their expression to the half
> du <- aspliDUexample1()
> duFilt <- filterDU( du, 'genes', '0.05', -1, absLogFC = FALSE,
                    logFCgreater = FALSE )
```

### 3.8.4 Subsetting ASpli objects

`ASpliCounts`, `ASpliAS`, `targets data.frame` and list of `GAlignments` (i.e. bam files data) can be subsetted by conditions or samples.

```
> # Subset counts
> counts <- aspliCountsExample()
> countsSmall <- subset(counts, targets, select = c("A_C", "A_D") )
> # Subset AS results. Note that PIR/PSI metrics are not recalculated.
> as <- aspliASexample()
> asSmall <- subset(as, targets, select = c("A_C_0", "A_C_1") )
> # Subset targets
> targets <- aspliTargetsExample()
> targetsSmall <- subsetTargets( targets, c("A_C", "A_C") )
```

```
> # Subset BAMs
> bams <- aspliBamsExample()
> bamSmall <- subsetBams( bams, targets, c("A_C", "A_C") )
```

### 3.8.5 Merging DU and junction data for bins

`mergeBinDUAS` function merge bin data from DU analysis contained in an `ASpliDU` object and bin data from junction repertoire usage and PSI/PIR metrics contained in an `ASpliAS` object. Also a delta PSI/PIR value is calculated from a contrast vector.

```
> du <- aspliDUexample1()
> as <- aspliASexample()
> mergeBinDUAS(du,as, targets)
```

## 4 Example data

### 4.0.1 Synthetic data set

ASpli contains a small synthetic data set to test the complete working pipeline. It consists of a genome with ten genes with multiple isoforms, a set of **BAM** files from an experiment with two×two factorial design, and a **GTF** file with the corresponding annotation. The two factors in this example are called **f1** and **f2**. **f1** can have value **A** or **B** and **f2** can have value **C** or **D**, defining four conditions: **A.C**, **A.D**, **B.C** and **B.D**. There are three replicates for each condition (see table 2).

Sample	f1	f2	replicate	condition
1	A	C	0	A.C
2	A	C	1	A.C
3	A	C	2	A.C
4	A	D	0	A.D
5	A	D	1	A.D
6	A	D	2	A.D
7	B	C	0	B.C
8	B	C	1	B.C
9	B	C	2	B.C
10	B	D	0	B.D
11	B	D	1	B.D
12	B	D	2	B.D

Table 2: Experimental design of the example data set

The first step of the workflow is to load the annotation from the reference sequence and extract their features.

```
> library( GenomicFeatures )
> # gtfFileName contains the full path to the example gtf file in your system.
> gtfFileName <- aspliExampleGTF()
```

## ASpli: An integrative R package for analysing alternative splicing using RNA-Seq

```
> # Create a TxDb object using makeTxDbFromGFF function from GenomicFeatures
> # package
> genomeTxDb <- makeTxDbFromGFF( gtfFileName )
> # Extract the genomic features
> features <- binGenome( genomeTxDb )
>
```

Also, the experimental sequencing data must be loaded, and the experimental samples, and factors should be defined.

```
> # bamFiles contains the full path of the bam files in your system
> bamFiles <- aspliExampleBamList( )
> # Define the targets
> targets <- data.frame(
  row.names = paste0('Sample',c(1:12)),
  bam = bamFiles,
  f1 = c( 'A','A','A','A','A','A',
          'B','B','B','B','B','B'),
  f2 = c( 'C','C','C','D','D','D',
          'C','C','C','D','D','D'),
  stringsAsFactors = FALSE
)
> # Examine the condition names
> getConditions(targets)
> # Load the bam files
> bams = loadBAM(targets)
```

After the targets are defined and the sequence data is read, you can continue to assign the number of counts to each feature.

```
> counts <- readCounts( features, bams, targets, readLength = 100,
  maxISize = 5000 )
```

Now you can run the DE/DU analysis, here will be evaluated a test to hypothesis that the expression of some genes respond to the interaction of **f1** and **f2**. In other words, that the observed effect of applying the two factors can not be explained by the independent contribution of each factor. The interaction can be expressed as:

$$I = (B.D - B.C) - (A.D - A.C) \quad 2$$

The coefficients of the terms in this expression can be represented in the contrast vector  $[1, -1, -1, 1]$ , assuming that they correspond to the order given by `getConditions` function.

```
> # DU/DE analysis for genes and bins.
> du <- DUreport( counts, targets, contrast = c( 1, -1, -1, 1 ) )
> # DU analysis for junctions
> du <- junctionDUreport( counts, targets, appendTo = du,
  contrast = c( 1, -1, -1, 1 ) )
```

The discovery of AS events can be done with:

```
> # AS analysis
> as <- AsDiscover( counts, targets, features, bams, readLength = 100)
```

## ASpli: An integrative R package for analysing alternative splicing using RNA-Seq

Finally, you can select your top tags integrating differential usage data and junction data.

```
> # Select top tags from DU
> topTagsBins <- filterDU( du, what = 'bins', fdr = 0.05, logFC = log(1.5,2) )
> # DU results are merged with AS results
> topTagsBins <- mergeBinDUAS( du, as, targets, contrast = c( 1, -1, -1, 1 ) )
> # We filter those top tags that have also junction evidence that supports the
> # differential usage, a change of at least 0.1 between observed and expected
> # in the PSI or PIR metric is required.
> topTagsBins <- topTagsBins[ abs(topTagsBins[, 'delta' ]) > 0.10, ]
```

Bin **GENE10:E002** is the one with the lowest **FDR** value, now we can examine it in a plot. The results in figure 9 show that a change in **f1** from **A** to **B** has not the same outcome for **C** or **D** in **f2**.

```
> plotGenomicRegions( features, 'GENE10:E002', genomeTxDb, targets )
```

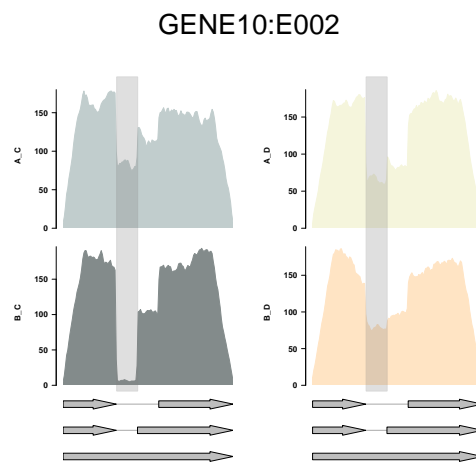


Figure 9: Genomic region of example bin GENE10:E002

### 4.0.2 Real data set

It is possible to run a demo of ASpli using public RNA-Seq data available via Bioconductor. It requires installing of `RNAseqData.HNRNPC.bam.chr14` package and loading an small TranscriptDb of Human Genome only for chromosome 14. This subset of reads is part of an experiment intended to analyze splicing factors and their relationship with exonization of Alu elements [10]. We load the package and the genome:

```
> library(RNAseqData.HNRNPC.bam.chr14)
```

In this case we are loading a genome `TranscriptDb` already created and available in the ASpli package for the *Homo sapiens* chromosome 14:

```
> chr14 <- system.file("extdata", "chr14.sqlite", package="ASpli")
> genome <- loadDb(chr14)
```

## ASpli: An integrative R package for analysing alternative splicing using RNA-Seq

And now we are ready to start the complete analysis.

```
> features <- binGenome(genome)
```

This data set contains data for of two conditions **KD** and **CT** with four replicates each one. Create targets object using the information of the available files:

```
> targets <- data.frame(  
  bam = RNAseqData.HNRNPC.bam.chr14_BAMFILES,  
  treat = c( "CT", "CT", "CT", "CT",  
             "KD", "KD", "KD", "KD" ),  
  stringsAsFactors = FALSE )
```

Then, load the bam files:

```
> bam <- loadBAM(targets)
```

Overlap alignments against features:

```
> counts <- readCounts( features, bam, targets, readLength=100L, maxISize=50000 )
```

Perform DE/DU analysis

```
> du <- DUreport(counts, targets)
```

Analyze AS using junctions:

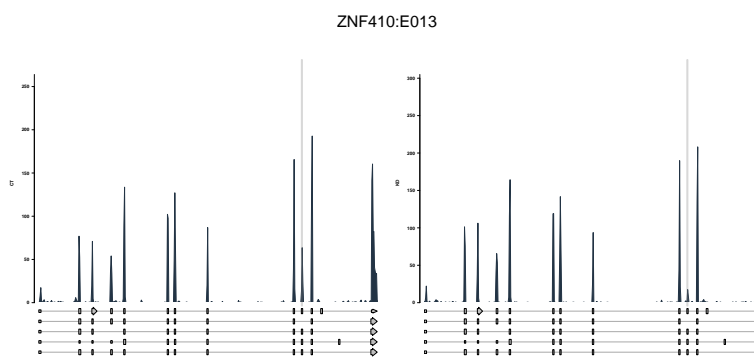
```
> as <- AsDiscover (counts, targets, features, bam, readLength=100L, threshold = 5, cores = 1 )
```

Select top tags, using DU and AS information:

```
> duFiltered <- filterDU( du, 'bins', fdr = 0.05, logFC = log(1.5,2) )  
> merged <- mergeBinDUAS( duFiltered, as, targets )
```

Bin ZNF410:E013, from top tags, has a usage change that is statistically significant and the change of PSI/PIR metric is greater than 0.3. We can inspect it visually, see figure 10.

```
> # Show the complete gene, with the bin highlighted  
> plotGenomicRegions( features, 'ZNF410:E013', genome, targets, verbose=TRUE,  
  colors = matrix(c('#223344','#223344'), ncol=2 ), outfileType='pdf', outfolder='grplots'  
> # Show a zoom in in the bin  
> plotGenomicRegions( features, 'ZNF410:E013', genome, targets, verbose=TRUE,  
  colors = matrix(c('#223344','#223344'), ncol=2 ),  
  zoomOnBins = 0.05)
```



**Figure 10:** Plot of ZNF410:E013 exon skipping

## References

- [1] G. P. Alamancos, E. Agirre, and E. Eyras. Methods to study splicing from high-throughput RNA sequencing data. *Methods in Molecular Biology*, 1126, 2014.
- [2] S. Anders, A. Reyes, and W. Huber. Detecting differential usage of exons from RNA-seq data. *Genome Research*, 22(10), 2012.
- [3] Xin Hong, Douglas G. Scofield, and Michael Lynch. Intron size, abundance, and distribution within untranslated regions of genes. *Molecular Biology and Evolution*, 23(12):2392–2404, 2006.
- [4] U. Braunschweig, N. L. Barbosa-Morais, Q. Pan, E. N. Nachman, B. Alipanahi, T. Gonatopoulos-Pournatzis, B. Frey, M. Irimia, and B. J. Blencowe. Widespread intron retention in mammals functionally tunes transcriptomes. *Genome Research*, 24(11), 2014.
- [5] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, Aug 2009. [PubMed Central:PM2723002] [DOI:10.1093/bioinformatics/btp352] [PubMed:19505943].
- [6] Y. Marquez, M. Hopfler, Z. Ayatollahi, A. Barta, and M. Kalyna. Unmasking alternative splicing inside protein-coding exons defines exitrons and their role in proteome plasticity. *Genome Research*, 25(7), 2015.
- [7] Y. Katz, E. T. Wang, E. M. Airoidi, and C. B. Burge. Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature Methods*, 7(12), 2010.
- [8] M. D. Robinson, D. J. McCarthy, and G. K. Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1), 2010.
- [9] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. Roy. Statist. Soc. Ser.*, (1), 1995.

- [10] Kathi Zarnack, Julian Konig, Mojca Tajnik, Iñigo Martincorena, Sebastian Eustermann, Isabelle Stevant, Alejandro Reyes, Simon Anders, Nicholas M. Luscombe, and Jernej Ule. Direct competition between hnrnp c and {U2AF65} protects the transcriptome from the exonization of alu elements. *Cell*, 152(3), 2013.