

Overlap encodings

Hervé Pagès

Last modified: May 2012; Compiled: August 27, 2012

WARNING (May 8, 2012): This vignette is incomplete and was still a WORK IN PROGRESS at the time `GenomicRanges` 1.8 was released (as part of Bioconductor 2.10, released in April 2012). All the work on *overlap encodings* is now happening in the **devel** version of the `GenomicRanges` package (`GenomicRanges` 1.9, part of Bioconductor 2.11, at the time of this writing). To use *overlap encodings* and related tools, and to get an updated version of this vignette, it is **strongly** recommended that you use Bioconductor 2.11. In particular please make sure that you always use the **latest** version of the `GenomicRanges` package (version 1.9.14 at the time of this writing, expect frequent updates).

In other words, this vignette is superseded by the vignette found in `GenomicRanges` 1.9 (part of Bioconductor 2.11). It won't be updated.

See `?useDevel` in the `BiocInstaller` package for how to use the devel version of Bioconductor (2.11 at the time of this writing).

See `?biocLite` in the `BiocInstaller` package for how to update all the installed packages.

Contents

1	Introduction	1
2	Example 1: With single-end reads	1
2.1	Load the reads and transcripts	1
2.2	Find and encode the overlaps	3
2.3	Detect overlaps showing “compatibility” with the transcript	4
2.4	Detect overlaps showing “almost compatibility” with the transcript	6
2.5	Combining results of <code>isCompatibleWithSplicing()</code> and <code>isCompatibleWithSkippedExons()</code> to detect novel splice junctions	7
3	Example 2: With paired-end reads	8
3.1	Load the reads and transcripts	8
3.2	Find and encode the overlaps	10
3.3	Detect overlaps showing “compatibility” with the transcript	11
4	<code>sessionInfo()</code>	12

1 Introduction

In the context of an RNA-seq experiment, encoding the overlaps between the aligned reads and the transcripts can be used for detecting those overlaps that are “compatible” with the splicing of the transcript.

Various tools are provided in the `IRanges` and `GenomicRanges` packages for working with *overlap encodings*. In this vignette, we illustrate the use of these tools on real RNA-seq data.

2 Example 1: With single-end reads

2.1 Load the reads and transcripts

We start by loading some aligned reads into a *GappedAlignments* object. The reads are stored in a BAM file (`untreated1_chr4.bam`) located in the `pasillaBamSubset` data package. This file contains single-end reads from an RNA-seq

experiment and aligned against the dm3 genome (see `?untreated1_chr4` in the `pasillaBamSubset` package for more information about those reads):

```
> library(pasillaBamSubset)
> untreated1_chr4()
```

```
[1] "E:/biocbld/bbs-2.10-bioc/R/library/pasillaBamSubset/extdata/untreated1_chr4.bam"
```

We use the `readGappedAlignments` function defined in the `GenomicRanges` package to read the BAM file into a *GappedAlignments* object. It's probably a good idea to get rid of the PCR or optical duplicate (flag bit 0x400 in the SAM format, see the SAM Spec ¹ for the details), as well as reads not passing quality controls (flag bit 0x200 in the SAM format). We do this by creating a *ScanBamParam* object that we pass to `readGappedAlignments` (see `?ScanBamParam` in the `Rsamtools` package for the details). Note that we also use `use.names=TRUE` in order to load the *query template names* (QNAME field in the SAM format) from the BAM file (`readGappedAlignments` will use them to set the names of the returned object):

```
> library(GenomicRanges)
> library(Rsamtools)
> flag0 <- scanBamFlag(isDuplicate=FALSE, isValidVendorRead=TRUE)
> param0 <- ScanBamParam(flag=flag0)
> gal14 <- readGappedAlignments(untreated1_chr4(), use.names=TRUE, param=param0)
```

Our reads can have up to 2 gaps (a gap corresponds to an N operation in the CIGAR):

```
> head(gal14)
```

GappedAlignments with 6 alignments and 0 elementMetadata cols:

	seqnames	strand	cigar	qwidth	start	end	width	ngap
	<Rle>	<Rle>	<character>	<integer>	<integer>	<integer>	<integer>	<integer>
SRR031729.3941844	chr4	-	75M	75	892	966	75	0
SRR031728.3674563	chr4	-	75M	75	919	993	75	0
SRR031729.8532600	chr4	+	75M	75	924	998	75	0
SRR031729.2779333	chr4	+	75M	75	936	1010	75	0
SRR031728.2826481	chr4	+	75M	75	949	1023	75	0
SRR031728.2919098	chr4	-	75M	75	967	1041	75	0

seqlengths:

chr2L	chr2R	chr3L	chr3R	chr4	chrM	chrX	chrYHet
23011544	21146708	24543557	27905053	1351857	19517	22422827	347038

```
> table(ngap(gal14))
```

0	1	2
184039	20169	147

We also need to retrieve the dm3 transcripts and their exons from UCSC, and extract the exons grouped by transcript in a *GRangesList* object. IMPORTANT: The reference genome of the transcripts must be **exactly** the same as the reference genome used to align the reads (note that this is a general rule, not only when working with overlap encodings):

```
> library(GenomicFeatures)
> dm3_refGene_txdb <- makeTranscriptDbFromUCSC(genome="dm3", tablename="refGene")
> exbytx <- exonsBy(dm3_refGene_txdb, by="tx")
```

We check that all the exons in any given transcript belong to the same chromosome and strand. Knowing that our set of transcripts is free of this kind of trans-splicing events will allow us some significant simplifications during the downstream analysis ². A quick and easy way to check this is to take advantage of the fact that `seqnames` and `strand` return *RleList* objects. So we can extract the number of Rle runs for each transcript and make sure it's always 1:

```
> table(elementLengths(runLength(seqnames(exbytx))))
```

¹<http://samtools.sourceforge.net/>

²Dealing with trans-splicing events is not covered in this document

```

1
26702
> table(elementLengths(runLength(strand(exbytx))))

```

```

1
26702

```

Therefore the strand of any given transcript is unambiguously defined and can be extracted with:

```
> exbytx_strand <- unlist(runValue(strand(exbytx)), use.names=FALSE)
```

2.2 Find and encode the overlaps

We are ready to compute the overlaps with the `findOverlaps` function. Note that the strand of the queries produced by the RNA-seq experiment is typically unknown so we use `ignore.strand=TRUE`:

```
> ov14 <- findOverlaps(gal14, exbytx, ignore.strand=TRUE)
```

However, the *overlap encodings* are strand sensitive so we will compute them twice, once for the original alignments (i.e. the alignments of the original queries), and once again for the “flipped alignments” (i.e. the alignments of the flipped queries). We extract the ranges of the original and flipped alignments in 2 *GRangesList* objects with:

```
> grl14o <- grglist(gal14, order.as.in.query=TRUE)
> grl14f <- flipQuery(grl14o)
```

and encode their overlaps with the transcripts:

```
> ovenc14o <- encodeOverlaps(grl14o, exbytx, hits=ov14)
> ovenc14f <- encodeOverlaps(grl14f, exbytx, hits=ov14)
```

`ovenc14o` and `ovenc14f` are 2 *OverlapsEncodings* objects of the same length as *Hits* object `ov14`. For each hit in `ov14`, we have 2 corresponding encodings, one in `ovenc14o` and one in `ovenc14f`, but only one of them encodes a hit between alignment ranges and exon ranges that are on the same strand. We use the `selectEncodingWithCompatibleStrand` function to merge them into a single *OverlapsEncodings* of the same length. For each hit in `ov14`, this selects the encoding corresponding to alignment ranges and exon ranges with compatible strand:

```
> grl14o_strand <- unlist(runValue(strand(grl14o)), use.names=FALSE)
> ovenc14 <- selectEncodingWithCompatibleStrand(ovenc14o, ovenc14f,
+                                              grl14o_strand, exbytx_strand,
+                                              hits=ov14)
> ovenc14
```

OverlapEncodings of length 617431

	Loffset	Roffset	encoding
[1]	0	3	1:i:
[2]	3	0	2:jm:am:
[3]	3	0	2:jm:am:
[4]	3	0	2:jm:am:
[5]	3	0	2:jm:am:
[6]	3	0	2:jm:am:
[7]	3	0	2:jm:am:
[8]	3	0	2:jm:am:
[9]	3	0	2:jm:am:
...
[617423]	24	0	1:i:
[617424]	23	0	1:i:
[617425]	23	0	1:i:
[617426]	24	0	1:i:
[617427]	22	0	1:i:
[617428]	24	0	1:i:
[617429]	23	0	1:i:
[617430]	23	0	1:i:
[617431]	24	0	1:i:

As a convenience, the 2 above calls to `encodeOverlaps` + merging step can be replaced by a single call to `encodeOverlaps` on either `grl14f` or `grl14o` with `flip.query.if.wrong.strand=TRUE`:

```
> ovenc14_again <- encodeOverlaps(grl14o, exbytx, hits=ov14, flip.query.if.wrong.strand=TRUE)
> stopifnot(identical(ovenc14_again, ovenc14))
```

Unique encodings in `ovenc14`:

```
> unique_ovenc14 <- levels(encoding(ovenc14))
> length(unique_ovenc14)

[1] 118

> head(unique_ovenc14)

[1] "1:c:" "1:d:" "1:e:" "1:f:" "1:h:" "1:i:"

> ovenc14_table <- table(encoding(ovenc14))
> tail(sort(ovenc14_table))

1:f: 1:k:c: 1:k: 1:c: 2:jm:af: 1:i:
1671 1933 14191 17732 82461 487756
```

Encodings are sort of cryptic but utilities are provided to extract specific meaning from them. Use of these utilities is covered in the next three subsections.

2.3 Detect overlaps showing “compatibility” with the transcript

We are interested in a particular type of overlap where the read overlaps the transcript in a “compatible” way, that is, in a way compatible with the splicing of the transcript. The `isCompatibleWithSplicing` function can be used on an *OverlapEncodings* object to detect this type of overlap. Note that `isCompatibleWithSplicing` can also be used on a character vector or factor.

`ovenc14` contains 7 unique encodings “compatible” with the splicing of the transcript:

```
> sort(ovenc14_table[isCompatibleWithSplicing(unique_ovenc14)])

2:jm:ag: 2:gm:af: 3:jmm:agm:aaf: 1:j: 1:f: 2:jm:af:
26 84 471 1607 1671 82461

1:i:
487756
```

Encodings "1:i:" (403826 occurrences in `ovenc14`), "2:jm:af:" (68914 occurrences in `ovenc14`), and "3:jmm:agm:aaf:" (438 occurrences in `ovenc14`), correspond to the following overlaps:

- "1:i:"


```
- read (no gap):      oooooooooo
- transcript:      ...  >>>>>>>>>>>>  ...
```
- "2:jm:af:"


```
- read (1 gap):      oooooo---ooo
- transcript:      ...  >>>>>>>>  >>>>>>>>  ...
```
- "3:jmm:agm:aaf:"


```
- read (2 gaps):      oo---ooooo---o
- transcript:      ...  >>>>>>>>  >>>>>  >>>>>>>>  ...
```

For clarity, only the exons involved in the overlap are represented. The transcript can of course have more upstream and downstream exons, which is denoted by the ... on the left side (5' end) and right side (3' end) of each drawing. Note that the exons represented in the 2nd and 3rd drawings are consecutive in the transcript.

Encodings "1:f:" and "1:j:" are variations of the situation described by encoding "1:i:". For "1:f:", the first aligned base of the read (or flipped read) is aligned with the first base of the exon. For "1:j:", the last aligned base of the read (or flipped read) is aligned with the last base of the exon:

```

• "1:f:"
  - read (no gap):      oooooooooo
  - transcript:      ...  >>>>>>>>>>>>  ...

• "1:j:"
  - read (no gap):      oooooooooo
  - transcript:      ...  >>>>>>>>>>>>  ...

> ov14_is_compat <- isCompatibleWithSplicing(ovenc14)
> table(ov14_is_compat) # 476124 "compatible" overlaps

ov14_is_compat
FALSE  TRUE
43355 574076

```

Number of “compatible” overlaps per alignment in gal14:

```

> gal14_ncompat <- tabulate(queryHits(ov14)[ov14_is_compat], nbins=length(gal14))
> elementMetadata(gal14)$ncompat <- gal14_ncompat
> head(gal14)

```

GappedAlignments with 6 alignments and 1 elementMetadata col:

	seqnames	strand	cigar	qwidth	start	end	width	ngap
	<Rle>	<Rle>	<character>	<integer>	<integer>	<integer>	<integer>	<integer>
SRR031729.3941844	chr4	-	75M	75	892	966	75	0
SRR031728.3674563	chr4	-	75M	75	919	993	75	0
SRR031729.8532600	chr4	+	75M	75	924	998	75	0
SRR031729.2779333	chr4	+	75M	75	936	1010	75	0
SRR031728.2826481	chr4	+	75M	75	949	1023	75	0
SRR031728.2919098	chr4	-	75M	75	967	1041	75	0

	ncompat
	<integer>
SRR031729.3941844	0
SRR031728.3674563	0
SRR031729.8532600	0
SRR031729.2779333	0
SRR031728.2826481	0
SRR031728.2919098	0

```

seqlengths:
  chr2L  chr2R  chr3L  chr3R  chr4  chrM  chrX  chrYHet
23011544 21146708 24543557 27905053 1351857 19517 22422827 347038

```

```

> table(gal14_ncompat)

gal14_ncompat
 0    1    2    3    4    5    6    7    8    9   10   11   12
57808 26660 23026 19728 14465 37356 9703  789 8881 1124  378 4389  48

```

Number of alignments in gal14 with at least 1 “compatible” overlap:

```

> sum(gal14_ncompat != 0)

[1] 146547

```

Number of “compatible” overlaps per transcript in exbytx:

```

> exbytx_ncompat14 <- tabulate(subjectHits(ov14)[ov14_is_compat], nbins=length(exbytx))
> names(exbytx_ncompat14) <- names(exbytx)
> tail(table(exbytx_ncompat14))

exbytx_ncompat14
11365 25384 33579 33936 40247 40400
 1    1    1    1    1    1

```

2.4 Detect overlaps showing “almost compatibility” with the transcript

In many aspects, “compatible” overlaps correspond to an ideal situation but in practise many overlaps don’t fall into that category. Here we are interested in a less perfect type of overlap where the read overlaps the transcript in a way that *would* be “compatible” if 1 or more exons were removed from the transcript. In that case we say that the overlap is “almost compatible” with the transcript. The `isCompatibleWithSkippedExons` function can be used on an *OverlapEncodings* object to detect this type of overlap. Note that `isCompatibleWithSkippedExons` can also be used on a character vector of factor.

`ovenc14` contains 7 unique encodings “almost compatible” with the splicing of the transcript:

```
> sort(ovenc14_table[isCompatibleWithSkippedExons(unique_ovenc14)])
```

```
2:jm:am:am:am:am:af: 2:jm:am:am:am:am:am:af: 2:gm:am:af: 2:jm:am:am:am:am:af:
1 1 3 7
3:jmm:agm:aam:aam:aaf: 3:jmm:agm:aam:aaf: 2:jm:am:am:af: 2:jm:am:af:
9 18 142 899
```

Encodings "2:jm:am:af:" (696 occurrences in `ovenc14`), "2:jm:am:am:af:" (114 occurrences in `ovenc14`), and "3:jmm:agm:aam:aaf:" (18 occurrences in `ovenc14`), correspond to the following overlaps:

- "2:jm:am:af:"

```
- read (1 gap):      ooooo-----ooo
- transcript:      ... >>>>>> >>>> >>>>>>> ...
```

- "2:jm:am:am:af:"

```
- read (1 gap):      ooooo-----ooo
- transcript:      ... >>>>>> >>>> >>>> >>>>>>> ...
```

- "3:jmm:agm:aam:aaf:"

```
- read (2 gaps):      oo----oooo-----oo
- transcript:      ... >>>>>> >>>> >>>> >>>>>>> ...
```

```
> ov14_is_almostcompat <- isCompatibleWithSkippedExons(ovenc14)
> table(ov14_is_almostcompat) # 837 "almost compatible" overlaps
```

```
ov14_is_almostcompat
FALSE TRUE
616351 1080
```

Number of “almost compatible” overlaps per alignment in `gal14`:

```
> gal14_nalmostcompat <- tabulate(queryHits(ov14)[ov14_is_almostcompat], nbins=length(gal14))
> elementMetadata(gal14)$nalmostcompat <- gal14_nalmostcompat
> head(gal14)
```

GappedAlignments with 6 alignments and 2 elementMetadata cols:

	seqnames	strand	cigar	qwidth	start	end	width	ngap
	<Rle>	<Rle>	<character>	<integer>	<integer>	<integer>	<integer>	<integer>
SRR031729.3941844	chr4	-	75M	75	892	966	75	0
SRR031728.3674563	chr4	-	75M	75	919	993	75	0
SRR031729.8532600	chr4	+	75M	75	924	998	75	0
SRR031729.2779333	chr4	+	75M	75	936	1010	75	0
SRR031728.2826481	chr4	+	75M	75	949	1023	75	0
SRR031728.2919098	chr4	-	75M	75	967	1041	75	0
	ncompat	nalmostcompat						
	<integer>	<integer>						
SRR031729.3941844	0	0						
SRR031728.3674563	0	0						

```

SRR031729.8532600      0      0
SRR031729.2779333      0      0
SRR031728.2826481      0      0
SRR031728.2919098      0      0

```

seqlengths:

```

chr2L chr2R chr3L chr3R chr4 chrM chrX chrYHet
23011544 21146708 24543557 27905053 1351857 19517 22422827 347038

```

```
> table(gal14_nalmostcompat)
```

```

gal14_nalmostcompat
 0    1    2    3    4    5    6    7    8    9   11
203851 291  49  96  22  16  11   3   4   8   4

```

Number of alignments in `gal14` with at least 1 “almost compatible” overlap:

```
> sum(gal14_nalmostcompat != 0)
```

```
[1] 504
```

Number of “almost compatible” overlaps per transcript in `exbytx`:

```

> exbytx_nalmostcompat14 <- tabulate(subjectHits(ov14)[ov14_is_almostcompat], nbins=length(exbytx))
> names(exbytx_nalmostcompat14) <- names(exbytx)
> table(exbytx_nalmostcompat14)

```

```

exbytx_nalmostcompat14
 0    1    2    3    4    6    7    8    9   10   12   13   14   18   20   21
26572  50   8  15  11   3   5   5  12   3   1   1   1   2   1   3
 32  34  44  59  77 170
  2   1   3   1   1   1

```

2.5 Combining results of `isCompatibleWithSplicing()` and `isCompatibleWithSkippedExons()` to detect novel splice junctions

An alignment in `gal14` with “almost compatible” hits but no “compatible” hits suggests the presence of one or more transcripts that are not in our annotations.

First we extract the index of those alignments:

```

> aln_shows_nov_splice_jct <- gal14_nalmostcompat != 0L &
+                               gal14_ncompat == 0L
> head(which(aln_shows_nov_splice_jct))

```

```
[1] 57972 57974 58321 67251 67266 67267
```

We make this an index into `ov14` (Hits object):

```
> is_nov_splice_jct <- queryHits(ov14) %in% which(aln_shows_nov_splice_jct)
```

We intersect with `is_almost_compat` to keep only the overlaps of interest:

```
> is_nov_splice_jct <- is_nov_splice_jct & ov14_is_almostcompat
```

For each overlap of interest, we extract the ranks of the skipped exons (we use a list for this as there might be more than 1 skipped exon per overlap):

```

> skpexrk <- extractSkippedExonRanks(ovenc14)[is_nov_splice_jct]
> table(elementLengths(skpexrk))

```

```

 1  2  3  4  5
242 123  7  1  1

```

Finally, we split `skpexrk` by transcript TxDb internal id:

```
> names(skpexrk) <- queryHits(ov14)[is_nov_splice_jct]
> f <- names(exbytx)[subjectHits(ov14)[is_nov_splice_jct]]
> tx2skpexrk <- split(skpexrk, f)
```

`tx2skpexrk` is a named list of named lists of integer vectors. The first level of names (outer names) are transcript TxDb internal ids and the second level of names (inner names) are alignment indices into `gal14`:

```
> head(names(tx2skpexrk)) # transcript TxDb internal ids
[1] "1"   "10"  "106" "107" "116" "117"
```

Transcript "10" has 7 hits. All of them skip exons 9 and 10:

```
> tx2skpexrk[["10"]]
```

```
$`104549`
[1] 9 10
```

```
$`104550`
[1] 9 10
```

```
$`104553`
[1] 9 10
```

```
$`104557`
[1] 9 10
```

```
$`104560`
[1] 9 10
```

```
$`104572`
[1] 9 10
```

```
$`104577`
[1] 9 10
```

Transcript "58" has 4 hits. Two of them skip exon 2, one of them skips exons 2 to 6, and one of them skips exon 10:

```
> tx2skpexrk[["58"]]
```

```
NULL
```

A few words about the interpretation of `tx2skpexrk`: Because of how we've conducted this analysis, the alignments reported in `tx2skpexrk` are guaranteed to NOT have any "compatible" overlaps with other known transcripts. All we can say, for example in the case of transcript "10", is that the 7 reported hits that skip exons 9 and 10 show evidence of one or more unknown transcripts with a splice junction that corresponds to the gap between exons 8 and 11. But without further analysis, we can't make any assumption about the exons structure of those unknown transcripts. In particular, we cannot assume the existence of an unknown transcript made of the same exons as transcript "10" minus exons 9 and 10!

3 Example 2: With paired-end reads

3.1 Load the reads and transcripts

We start by loading some aligned paired-end reads into a *GappedAlignmentPairs* object. The reads are stored in a BAM file (`untreated3_chr4.bam`) located in the `pasillaBamSubset` data package. This file contains paired-end reads from an RNA-seq experiment and aligned against the dm3 genome (see `?untreated3_chr4` in the `pasillaBamSubset` package for more information about those reads):


```
> untreated3_chr4()
```

```
[1] "E:/biocbld/bbs-2.10-bioc/R/library/pasillaBamSubset/extdata/untreated3_chr4.bam"
```

We use the `readGappedAlignmentPairs` function to read the BAM file into a *GappedAlignmentPairs* object:

```
> galp34 <- readGappedAlignmentPairs(untreated3_chr4(), use.names=TRUE, param=param0)
> head(galp34)
```

GappedAlignmentPairs with 6 alignment pairs and 0 elementMetadata cols:

	seqnames	strand	:	ranges	:	ranges
	<Rle>	<Rle>	:	<IRanges>	:	<IRanges>
SRR031715.1138209	chr4	+	:	[169, 205]	:	[326, 362]
SRR031714.756385	chr4	+	:	[943, 979]	:	[1086, 1122]
SRR031714.2355189	chr4	+	:	[944, 980]	:	[1119, 1155]
SRR031714.5054563	chr4	+	:	[946, 982]	:	[986, 1022]
SRR031715.1722593	chr4	+	:	[966, 1002]	:	[1108, 1144]
SRR031715.2202469	chr4	+	:	[966, 1002]	:	[1114, 1150]

seqlengths:

chr2L	chr2R	chr3L	chr3R	chr4	chrM	chrX	chrYHet
23011544	21146708	24543557	27905053	1351857	19517	22422827	347038

The `show` method for *GappedAlignmentPairs* objects displays two `ranges` columns, one for the *first* alignment in the pair (the left column), and one for the *last* alignment in the pair (the right column). The `strand` column corresponds to the strand of the *first* alignment.

```
> head(first(galp34))
```

GappedAlignments with 6 alignments and 0 elementMetadata cols:

	seqnames	strand		cigar	qwidth	start	end	width	ngap
	<Rle>	<Rle>	<character>	<integer>	<integer>	<integer>	<integer>	<integer>	<integer>
SRR031715.1138209	chr4	+		37M	37	169	205	37	0
SRR031714.756385	chr4	+		37M	37	943	979	37	0
SRR031714.2355189	chr4	+		37M	37	944	980	37	0
SRR031714.5054563	chr4	+		37M	37	946	982	37	0
SRR031715.1722593	chr4	+		37M	37	966	1002	37	0
SRR031715.2202469	chr4	+		37M	37	966	1002	37	0

seqlengths:

chr2L	chr2R	chr3L	chr3R	chr4	chrM	chrX	chrYHet
23011544	21146708	24543557	27905053	1351857	19517	22422827	347038

```
> head(last(galp34))
```

GappedAlignments with 6 alignments and 0 elementMetadata cols:

	seqnames	strand		cigar	qwidth	start	end	width	ngap
	<Rle>	<Rle>	<character>	<integer>	<integer>	<integer>	<integer>	<integer>	<integer>
SRR031715.1138209	chr4	-		37M	37	326	362	37	0
SRR031714.756385	chr4	-		37M	37	1086	1122	37	0
SRR031714.2355189	chr4	-		37M	37	1119	1155	37	0
SRR031714.5054563	chr4	-		37M	37	986	1022	37	0
SRR031715.1722593	chr4	-		37M	37	1108	1144	37	0
SRR031715.2202469	chr4	-		37M	37	1114	1150	37	0

seqlengths:

chr2L	chr2R	chr3L	chr3R	chr4	chrM	chrX	chrYHet
23011544	21146708	24543557	27905053	1351857	19517	22422827	347038

According to the SAM format specifications, the aligner is expected to mark each alignment pair as *proper* or not (flag bit 0x2 in the SAM format). The SAM Spec only says that a pair is *proper* if the *first* and *last* alignments in the pair are “properly aligned according to the aligner”. So the exact criteria used for setting this flag is left to the aligner.

We use `isProperPair` to extract this flag from the *GappedAlignmentPairs* object:

```
> table(isProperPair(galp34))
```

```
FALSE TRUE
29518 45828
```

Even though we could do *overlap encodings* with the full object, we keep only the *proper* pairs for our downstream analysis:

```
> galp34 <- galp34[isProperPair(galp34)]
```

For the transcript, we'll reuse the `exbytx` object obtained previously.

3.2 Find and encode the overlaps

Let's compute the overlaps:

```
> ov34 <- findOverlaps(galp34, exbytx, ignore.strand=TRUE)
```

and encode them:

```
> grl34 <- grglist(galp34, order.as.in.query=TRUE)
> ovenc34 <- encodeOverlaps(grl34, exbytx, hits=ov34, flip.query.if.wrong.strand=TRUE)
> ovenc34
```

```
OverlapEncodings of length 126337
      Loffset Roffset  encoding
[1]          4        0 1--1:i--k:
[2]          4        0 1--1:i--i:
[3]          4        0 1--1:i--k:
[4]          4        0 1--1:i--k:
[5]          4        0 1--1:i--m:
[6]          4        0 1--1:i--i:
[7]          3        1 1--1:i--i:
[8]          3        1 1--1:i--i:
[9]          2        2 1--1:i--i:
...          ...        ...
[126329]       24        0 1--1:i--i:
[126330]       23        0 1--1:i--i:
[126331]       23        0 1--1:i--i:
[126332]       24        0 1--1:i--i:
[126333]       22        0 1--1:i--i:
[126334]       24        0 1--1:i--i:
[126335]       23        0 1--1:i--i:
[126336]       23        0 1--1:i--i:
[126337]       24        0 1--1:i--i:
```

Unique encodings in `ovenc34`:

```
> unique_ovenc34 <- levels(encoding(ovenc34))
> length(unique_ovenc34)
```

```
[1] 132
```

```
> head(unique_ovenc34)
```

```
[1] "1--1:a--c:" "1--1:a--e:" "1--1:a--f:" "1--1:a--i:" "1--1:a--j:" "1--1:a--k:"
```

```
> ovenc34_table <- table(encoding(ovenc34))
> tail(sort(ovenc34_table))
```

```
1--1:i--k:      1--1:c--i:      1--1:i--m: 1--2:i--jm:a--af: 2--1:jm--m:af--i:
      1695              2090              2458              2500              2924
1--1:i--i:
      107102
```

3.3 Detect overlaps showing “compatibility” with the transcript

ovenc34 contains 13 unique encodings “compatible” with the splicing of the transcript:

```
> sort(ovenc34_table[isCompatibleWithSplicing(unique_ovenc34)])
```

1--2:f--jm:a--af:	1--1:f--j:	2--1:jm--m:af--j:
3	11	18
2--1:jm--m:af--f:	1--1:i--m:a--f:	1--1:j--m:a--i:
24	46	54
2--2:jm--mm:af--jm:aa--af:	1--1:i--m:a--i:	1--1:i--j:
138	371	433
1--1:f--i:	1--2:i--jm:a--af:	2--1:jm--m:af--i:
795	2500	2924
1--1:i--i:		
107102		

Encodings "1--1:i--i:" (89039 occurrences in `ovenc34`), "2--1:jm--m:af--i:" (2825 occurrences in `ovenc34`), "1--2:i--jm:a--af:" (2339 occurrences in `ovenc34`), and "1--1:i--m:a--i:" (342 occurrences in `ovenc34`), correspond to the following overlaps:

- "1--1:i--i:"
 - paired-end read (no gap on the first end, no gap on the last end):
oooo oooo
 - transcript: ... >>>>>>>>>>>>>>> ...
- "2--1:jm--m:af--i:"
 - paired-end read (1 gap on the first end, no gap on the last end):
oooo---o oooo
 - transcript: ... >>>>>>> >>>>>>>>>>> ...
- "1--2:i--jm:a--af:"
 - paired-end read (no gap on the first end, 1 gap on the last end):
oooo oo---oo
 - transcript: ... >>>>>>>>>>>>>>> >>>>>>>> ...
- "1--1:i--m:a--i:"
 - paired-end read (no gap on the first end, no gap on the last end):
oooo oooo
 - transcript: ... >>>>>>>> >>>>>>> ...

Note: switch use of “first” and “last” above if the read was flipped.

```
> ov34_is_compat <- isCompatibleWithSplicing(ovenc34)
> table(ov34_is_compat) # 95801 "compatible" overlaps
```

```
ov34_is_compat
FALSE      TRUE
11918 114419
```

Number of “compatible” overlaps per alignment pair in galp34:

```
> galp34_ncompat <- tabulate(queryHits(ov34)[ov34_is_compat], nbins=length(galp34))
> elementMetadata(galp34)$ncompat <- galp34_ncompat
> head(galp34)
```

GappedAlignmentPairs with 6 alignment pairs and 1 elementMetadata col:

	seqnames	strand	:	ranges	:	ranges		ncompat
	<Rle>	<Rle>	:	<IRanges>	:	<IRanges>		<integer>
SRR031715.1138209	chr4	+	:	[169, 205]	:	[326, 362]		0
SRR031714.756385	chr4	+	:	[943, 979]	:	[1086, 1122]		0
SRR031714.5054563	chr4	+	:	[946, 982]	:	[986, 1022]		0
SRR031715.1722593	chr4	+	:	[966, 1002]	:	[1108, 1144]		0
SRR031715.2202469	chr4	+	:	[966, 1002]	:	[1114, 1150]		0
SRR031714.3544437	chr4	-	:	[1087, 1123]	:	[963, 999]		0

seqlengths:

chr2L	chr2R	chr3L	chr3R	chr4	chrM	chrX	chrYHet
23011544	21146708	24543557	27905053	1351857	19517	22422827	347038

```
> table(galp34_ncompat)
```

galp34_ncompat												
0	1	2	3	4	5	6	7	8	9	10	11	12
15164	6348	5473	4706	1491	7456	1991	123	2201	105	69	699	2

Number of alignment pairs in galp34 with at least 1 “compatible” overlap:

```
> sum(galp34_ncompat != 0)
```

```
[1] 30664
```

Number of “compatible” overlaps per transcript in exbytx:

```
> exbytx_ncompat34 <- tabulate(subjectHits(ov34)[ov34_is_compat], nbins=length(exbytx))
> names(exbytx_ncompat34) <- names(exbytx)
> tail(table(exbytx_ncompat34))
```

exbytx_ncompat34						
2686	4886	5228	5246	7484	7490	
1	1	1	1	1	1	

4 sessionInfo()

```
> sessionInfo()
```

R version 2.15.1 (2012-06-22)

Platform: i386-pc-mingw32/i386 (32-bit)

locale:

```
[1] LC_COLLATE=C                      LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
[5] LC_TIME=English_United States.1252
```

attached base packages:

```
[1] stats      graphics  grDevices  utils      datasets  methods    base
```

other attached packages:

```
[1] pasillaBamSubset_0.0.2          BSgenome.Scerevisiae.UCSC.sacCer2_1.3.17
[3] org.Sc.sgd.db_2.7.1            RSQLite_0.11.1
[5] DBI_0.2-5                      GenomicFeatures_1.8.3
[7] AnnotationDbi_1.18.1          leeBamViews_0.99.19
[9] BSgenome_1.24.0               Biobase_2.16.0
[11] EatonEtAlChIPseq_0.0.7        rtracklayer_1.16.3
[13] ShortRead_1.14.4              latticeExtra_0.6-24
[15] RColorBrewer_1.0-5            lattice_0.20-10
```

```
[17] Rsamtools_1.8.6          Biostrings_2.24.1
[19] GenomicRanges_1.8.13     IRanges_1.14.4
[21] BiocGenerics_0.2.0
```

loaded via a namespace (and not attached):

```
[1] RCurl_1.91-1.1 XML_3.9-4.1    biomaRt_2.12.0 bitops_1.0-4.1 grid_2.15.1    hwriter_1.3
[7] stats4_2.15.1  tools_2.15.1  zlibbioc_1.2.0
```