Overview of svrearrange package

Robert Scharpf

August 31, 2017

Introduction

This package is used to identify somatic rearrangements.

Identifying candidate rearrangements

```r
library(svbams)
library(svfilters.hg19)
library(svcnvs)
```

```
## Loading required package: svclasses
```

```r
library(svalignments)
library(svrearrange)
```

This package uses improperly paired reads to identify candidate somatic rearrangements. We assume the improperly paired reads have been extracted from a bam file (see the `svalignments` package). Here, we load the improperly paired reads for the example BAM file available from an ovarian cell line in the R package `svbams`.

```r
extdata <- system.file("extdata", package="svbams")
id <- "cgov44t_revised.bam"
bam.file <- file.path(extdata, id)
irp.file <- file.path(extdata, "cgov44t_improper.rds")
irp <- readRDS(irp.file)
seqlevels(bins1kb, pruning.mode="coarse") <- paste0("chr", c(1:22, "X"))
seqlevels(irp, pruning.mode="coarse") <- seqlevels(bins1kb)
```

Below, we use simple helper functions to assemble different pieces of data needed for rearrangement analyses. In particular, we create a list of filters used to reduce the identification of spurious alignment artifacts. This `GRangesList` of filters is created by the function `rFilters` and includes somatic deletions and amplifications identified from read-depth analyses as part of the `svcnvs` vignette, outliers in read depth coverage and germline CNVs. Both the outliers and germline CNVs were identified from a set of 10 lymphoblastoid cell lines and germline CNVs and are provided by the `svfilters.hg19` package in the `germline_filters` object.

```r
data(amplicon_graph, package="svcnvs")
data(deletions2, package="svcnvs")
genome_filters <- reduceGenomeFilters(germline_filters,
                                      seqlevels(bins1kb))
filters <- rFilters(amplicons=ampliconRanges(amplicon_graph),
                    deletions=variant(deletions2),
                    rear=germline_rear,
                    germline=genome_filters)
rdat <- rearrangementData(bins=bins1kb,
                          read_pairs=list(improper=irp),
                          filters=filters)
```

The function `findCandidates2` identifies clusters of improperly paired reads. Parameters for identifying improperly paired read clusters are specified in the `RearrangementParams` function. In particular, note the `min_number_tags_per_cluster` argument. With 30x coverage and a clonal ovarian cancer cell line, we set this value to 5. Note, we require the read pairs to be at least 10kb apart with respect to the reference genome to reduce false positives.

```
rparam <- RearrangementParams(min_number_tags_per_cluster=5,
                              rp_separation=10e3)
minNumberTagsPerCluster(rparam)
```

```
## [1] 5
```

```
rpSeparation(rparam)
```

```
## [1] 10000
```

```
rlist <- findCandidates2(rdat, rparam)
rdat$rlist <- rlist
rlist
```

```
## An object of class 'RearrangementList'
##   number rearrangement objects:  2
##   Use '[[i]]' to return a single Rearrangement object'
```

Two candidate rearrangements are identified. Each improperly paired read flanking a candidate rearrangement is typed according to the orientation and spacing of the paired reads. See @TODO@ for more information on the rearrangement types. Each rearrangement, supported by at least 5 read pairs in this example, is classified according to the modal type. To extract all the improperly paired reads supporting the first rearrangement, one can use the function `improper`:

```
improper(rlist[[1]])
```

```
## GAlignmentPairs object with 52 pairs, strandMode=1, and 0 metadata columns:
##         seqnames strand   :                 ranges  --
##            <Rle>  <Rle>   :              <IRanges>  --
##   1-2     chr8      -     : [128691750, 128691849]  --
##   1-2     chr8      -     : [128691808, 128691907]  --
##   1-2     chr8      -     : [128691814, 128691913]  --
##   1-2     chr8      -     : [128691828, 128691927]  --
##   1-2     chr8      -     : [128691838, 128691937]  --
##   ...      ...    ... ...                    ... ...
##   1-2     chr8      +     : [129615488, 129615587]  --
##   1-2     chr8      +     : [129615488, 129615587]  --
##   1-2     chr8      +     : [129615489, 129615588]  --
##   1-2     chr8      +     : [129615495, 129615594]  --
##   1-2     chr8      +     : [129615503, 129615602]  --
##                       ranges
##                    <IRanges>
##   1-2 [129615393, 129615492]
##   1-2 [129615468, 129615567]
##   1-2 [129615443, 129615542]
##   1-2 [129615437, 129615536]
##   1-2 [129615382, 129615481]
##   ...                    ...
##   1-2 [128691798, 128691897]
##   1-2 [128691886, 128691985]
##   1-2 [128691929, 128692028]
##   1-2 [128691913, 128692012]
##   1-2 [128691839, 128691938]
##   -------
##   seqinfo: 23 sequences from an unspecified genome
```

The rearranged read pair clusters link potentially distant regions within a chromosome (intra-chromosome) or between chromosomes (inter-chromosomal). These links can be accessed by the `linkedBins` accessor:

```
linkedBins(rlist)
```

```
## GRanges object with 2 ranges and 1 metadata column:
##       seqnames                 ranges strand |               linked.to
##          <Rle>              <IRanges>  <Rle> |               <GRanges>
##   1-2     chr8 [128691748, 128692097]      * | chr8:129615326-129615637
##   3-4    chr15 [ 63093892,  63094115]      * |  chr15:63357756-63358139
##   -------
##   seqinfo: 23 sequences from an unspecified genome
```

The `linked.to` value in the `mcols` of the linked bins is itself a `GRanges` object:

```
linkedBins(rlist)$linked.to
```

```
## GRanges object with 2 ranges and 0 metadata columns:
##       seqnames                 ranges strand
##          <Rle>              <IRanges>  <Rle>
##   [1]     chr8 [129615326, 129615637]      *
##   [2]    chr15 [ 63357756,  63358139]      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

Next, we apply the filters we've assembled:

```
rlist <- filterRear(rdat, rparam)
```

Like `rlist`, `rlist2` is also a `RearrangementList` object.

Confirmation by BLAT

The rearangements identified thus far are required to have a physical separation of at least `rp_separation` with respect to the reference genome, must be supported by at least `min_number_tags_per_cluster` paired reads, at least `prop_modal_rearrangement` of the read pairs must be consistent with the modal rearrangement type, and the size of the read clusters must be at least `min_cluster_size` basepairs. This section demonstrates how realignment using BLAT can further improve specifity of the rearrangement analysis.

Mapped-mapped

For each rearrangement identified in the above analyses, we have saved the set of improperly paired reads supporting the new sequence junction. Because both reads in the improper pairs were mapped to the reference genome by ELAND, this section describes the realignment of 'mapped-mapped' pairs. In the following code chunk, we extract the tag sequence of all improperly paired reads in a `RearrangementList` and write these sequences to file in fasta format. Note, the `MAX` argument to `getSequenceOfReads` below indicates the maximum number of sequences for a specific rearrangement to extract from the BAM file. The first rearrangement in the `rlist` object has 52 improperly paired reads spanning the sequence junction. Setting this parameter to 25 (`MAX=25`), we randomly select 25 of the 52 read pairs for re-alignment by BLAT.

```
set.seed(123)
tags <- getSequenceOfReads(rlist, bam.file, MAX=25L, build = "hg19")
```

```
dir <- tempdir()
fa.file <- file.path(dir, paste0(id, ".fa"))
writeToFasta(tags, fa.file)
```

```
## [1] TRUE
```

The unevaluated code below illustrates how one could do a `system` call to run the command-line version of BLAT. In addition to requiring installation of BLAT, we must also have a copy of the appropriate reference genome available.

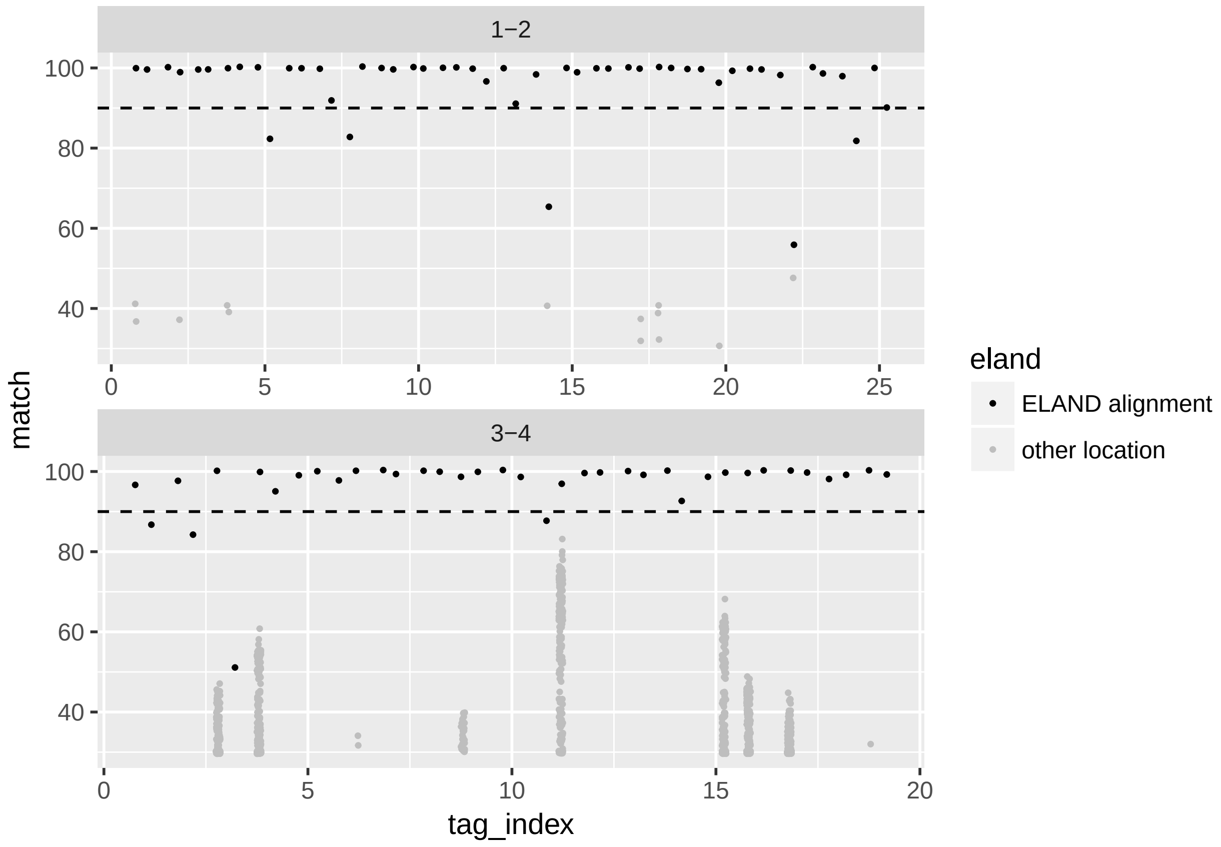
```
blatpath <- "~/bin/x86_64/blat"
refgenome <- "~/Dropbox/reference_genome/hg19.fa"
outfile <- tempfile()
cmd <- paste(blatpath, refgenome, fa.file, outfile)
system(cmd)
file.copy(outfile, "../../svalignments/inst/extdata/blat_alignment.txt")
```

Here, we read the previously saved BLAT alignments for this data included with the `svalignments` package.

```
extdata <- system.file("extdata", package="svalignments")
blat.file <- file.path(extdata, "blat_alignment.txt")
blat_aln <- readBlat(blat.file)
```

Next, we parse the BLAT alignments (@DESCRIBE@) and assess whether the BLAT alignment for each read pair is consistent with the whole genome aligner (we use ELAND). In particular, we require that each read have only one near perfect match in the genome (match > 90) and, if there is a near-perfect match, the near-perfect match must be consistent with the ELAND alignment. In the following code chunk, we visualize the BLAT scores for 25 improper read pairs for rearrangement `1-2` and the 19 improper read pairs for the `3-4` rearrangement.

```
scores <- blatScores(blat_aln, tags, id="CGOV44T", thr=0.8)
scores$rearrangement <- factor(scores$rearrangement)
scores$eland <- c("other location", "ELAND alignment")[as.integer(scores$is_overlap) + 1]
scores$eland <- factor(scores$eland)
library(ggplot2)
ggplot(scores, aes(tag_index, match)) +
  geom_jitter(width=0.05, aes(color=eland), size=0.5) +
  geom_hline(yintercept=90, linetype="dashed") +
  scale_color_manual(values=c("black", "gray")) +
  facet_wrap(~rearrangement, ncol=1, scales="free_x")
```



Mapped-unmapped

In addition to using BLAT to confirm the ELAND alignments of the improperly paired reads supporting a rearrangement, BLAT can also be used to identify split reads that directly span the sequence junction. Here, we query the BAM file for all read pairs in which a read was mapped near the putative rearrangement but its mate was not aligned (mapped-unmapped). If the unmapped mate spans the sequence junction, we will assess whether BLAT aligns a subsequence of the read to one cluster and the complement of the subsequence to the second cluster. Such reads further improve the specificity of the rearrangement and establish basepair resolution of the sequence junction. First, we construct a `GRanges` object of all the linked read clusters identified in our rearrangement analyses. This `GRanges` object will be used to query the BAM file for any read pair in which one read mapped to this region near the rearrangement and its mate did not. Next, we write the sequence of the unmapped reads to a fasta file.

```
query <- uncouple(linkedBins(rlist))
unmapped <- unmapped_read(bam.file, query, yield_size=200000)
```

```
## .
```

```
length(unmapped)
```

```
## [1] 87
```

```
dir <- tempdir()
mapped_unmapped.fa <- file.path(dir, "mapped-unmapped.fa")
writeUnmappedToFasta(unmapped, mapped_unmapped.fa)
```

The object `unmapped` is a `GRanges` object of 87 reads that were not mapped by ELAND to the reference genome and that have a mate mapped to one of the intervals in `query`. Again, we use a system call in the following unevaluated code chunk to realign the unmapped reads:

```
outfile <- tempfile()
cmd <- paste(blatpath, refgenome, mapped_unmapped.fa, outfile)
system(cmd)
file.copy(outfile, "../../svalignments/inst/extdata/blat_unmapped.txt")
```

TODO: create a script in data-raw/ that creates the above blat_unmapped.txt object.

```
unmap.file <- file.path(extdata, "blat_unmapped.txt")
blat_unmap <- readBlat(unmap.file)
```

Recall that our QC analysis of the mapped-mapped read pairs focused on whether the original ELAND alignment was the only high-scoring BLAT alignment. Here, our goal is to assess whether BLAT splits the alignment of the ELAND-unmapped reads to the approximate locations of the rearrangement given by the `rlist` object. To make this more concrete, the locations in the first rearrangement (with respect to the reference genome) that we think are joined in the somatic genome are given by the linked bins:

```
linkedBins(rlist)[1]
```

```
## GRanges object with 1 range and 1 metadata column:
##       seqnames                 ranges strand |                linked.to
##          <Rle>              <IRanges>  <Rle> |                <GRanges>
##   1-2     chr8 [128691748, 128692097]      * | chr8:129615326-129615637
##   -------
##   seqinfo: 23 sequences from an unspecified genome
```

We want to assess whether BLAT aligns part of a subsequence of a read to this region

```
granges(linkedBins(rlist)[1])
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames                 ranges strand
##          <Rle>              <IRanges>  <Rle>
##   1-2     chr8 [128691748, 128692097]      *
##   -------
##   seqinfo: 23 sequences from an unspecified genome
```

and the compliment of the above subsequence to this bin:

```
linkedTo(rlist)[1]
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames                 ranges strand
##          <Rle>              <IRanges>  <Rle>
##   [1]     chr8 [129615326, 129615637]      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

We use the function `rearrangedReads` to identify BLAT records that correspond to split reads supporting a rearrangement. Depending on the size of the sequenced DNA fragments, the improperly paired reads can only approximate the location of a sequence junction (likely to within 100 basepairs).

```
split_reads <- rearrangedReads(linkedBins(rlist), blat_unmap, 500)
elementNROWS(split_reads)
```

```
## 1-2 3-4
##  60  12
```

```
split_reads
```

```
## GRangesList object of length 2:
## $1-2
## GRanges object with 60 ranges and 8 metadata columns:
##        seqnames                 ranges strand |        revmap   qStarts
##           <Rle>              <IRanges>  <Rle> | <IntegerList> <integer>
##    [1]     chr8 [128691765, 128691802]      * |             2        62
##    [2]     chr8 [129615526, 129615594]      * |             1         0
##    [3]     chr8 [128691765, 128691836]      * |             1        28
##    [4]     chr8 [129615560, 129615594]      * |             2         0
##    [5]     chr8 [128691765, 128691827]      * |             1        37
##    ...      ...                    ...    ... .           ...       ...
##   [56]     chr8 [129615549, 129615594]      * |             2         0
##   [57]     chr8 [128691765, 128691801]      * |             2        63
##   [58]     chr8 [129615525, 129615594]      * |             1         0
##   [59]     chr8 [128691765, 128691810]      * |             2        54
##   [60]     chr8 [129615534, 129615594]      * |             1         0
##        blockSizes     match     rear.id                          qname
##         <integer> <numeric> <character>                    <character>
##    [1]         38        38         1-2  HS2000-887_357:3:1102:9785:14171
##    [2]         69        68         1-2  HS2000-887_357:3:1102:9785:14171
##    [3]         72        72         1-2 HS2000-887_357:3:1104:13640:37460
##    [4]         35        34         1-2 HS2000-887_357:3:1104:13640:37460
##    [5]         63        63         1-2 HS2000-887_357:3:1114:17233:88112
##    ...        ...       ...         ...                               ...
##   [56]         46        45         1-2  HS2000-887_357:5:2311:4506:57210
##   [57]         37        37         1-2  HS2000-887_357:5:2312:3646:81619
##   [58]         70        69         1-2  HS2000-887_357:5:2312:3646:81619
##   [59]         46        45         1-2  HS2000-887_357:5:2314:7153:74869
##   [60]         61        60         1-2  HS2000-887_357:5:2314:7153:74869
##         gapbases     Qsize
##        <integer> <integer>
##    [1]         0       100
##    [2]         0       100
##    [3]         0       100
##    [4]         0       100
##    [5]         0       100
##    ...       ...       ...
##   [56]         0       100
##   [57]         0       100
##   [58]         0       100
##   [59]         0       100
##   [60]         0       100
##
## ...
## <1 more element>
## -------
## seqinfo: 2 sequences from an unspecified genome; no seqlengths
splitReads(rlist) <- split_reads
```

5-prime to 3-prime orientation

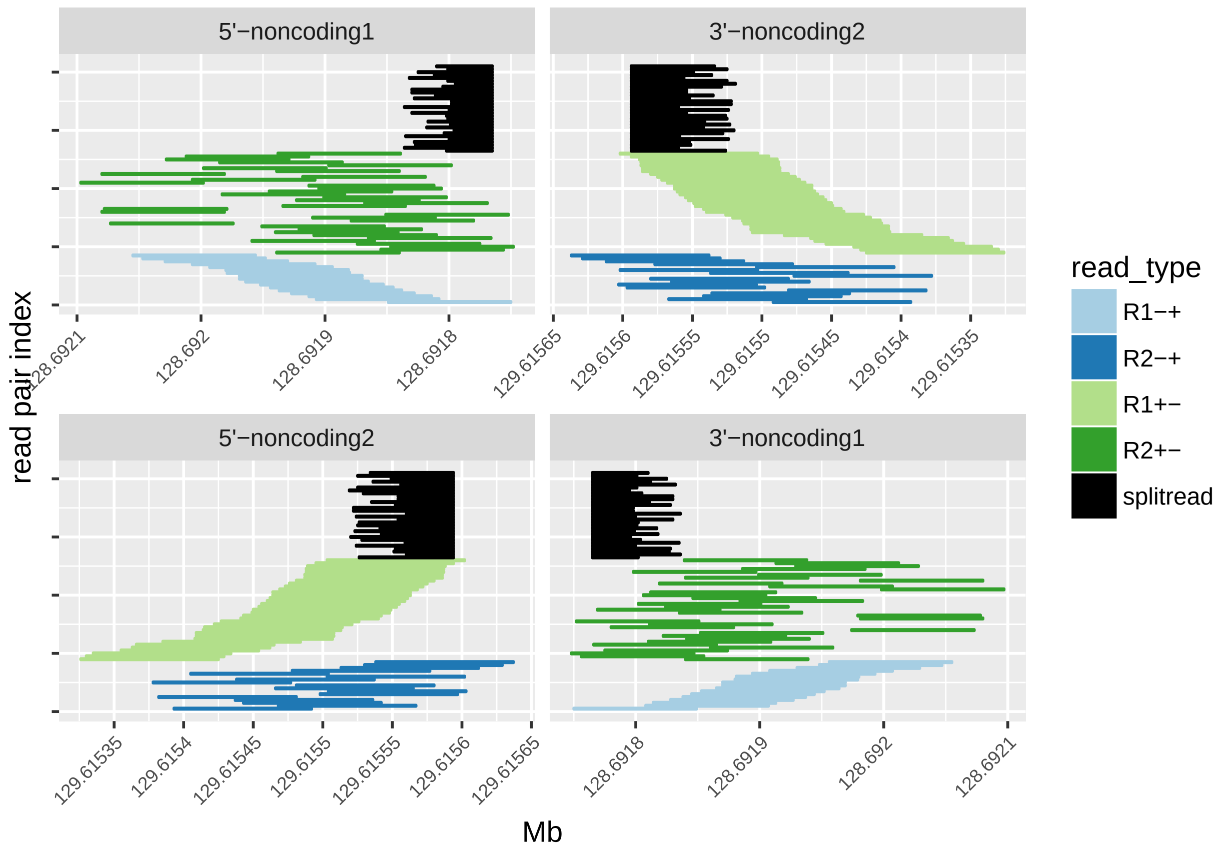
The above BLAT analysis identifies 60 and 12 split reads for the two rearrangements named `1-2` and `3-4`, respectively. Each of these rearrangements have two possible 5-prime to 3-prime orientations. The function `fiveTo3List` places the linked bins in their 5-prime to 3-prime orientation. Note, in the code below that `rlist2` is now twice the length of the original `rlist` object as each rearrangement has been placed in two possible orientations (This is required for evaluating in-frame fusions).

```
rlist2 <- fiveTo3List(rlist, build="hg19")
rlist2
```

```
## An object of class 'RearrangementList'
##   number rearrangement objects:  4
##   Use '[[i]]' to return a single Rearrangement object'
```

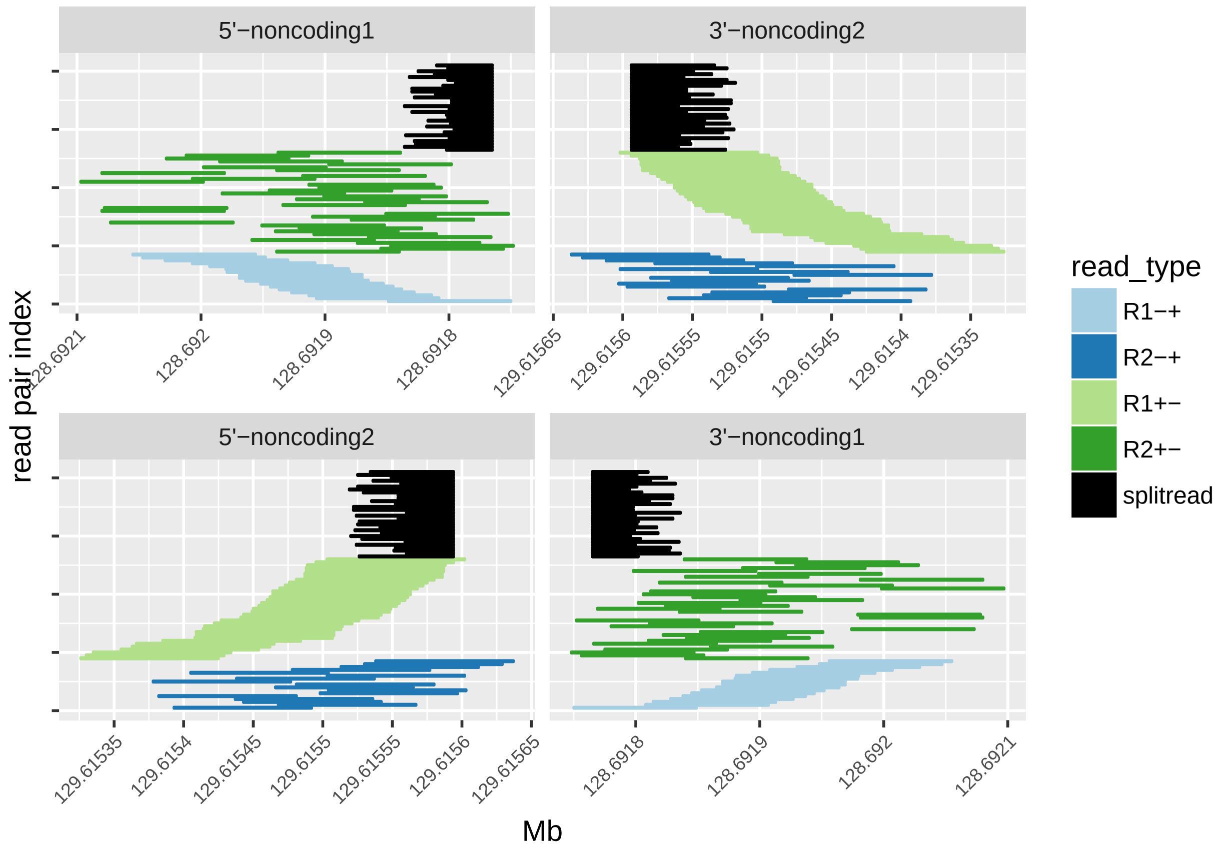
To visualize the improperly paired reads and the split reads supporting a rearrangement, we first collect the supporting reads in a `data.frame`. The function `rearDataFrameList` (need better name) takes a `RearrangementList` as input and extracts the supporting reads belonging to the *first two* elements of the list that correspond to the two possible 5-prime to 3-prime orientations. Next, we use `ggRearrange`, a wrapper to `ggplot`, to visualize the supporting reads. Because sequence junctions do not overlap a transcript, we've arbitarily labeled the two genomic regions that are joined in the somatic genome as `noncoding1` and `noncoding2`.

```
df <- rearDataFrameList(rlist2[1:2], build="hg19")
ggRearrange(df)
```



The second rearrangement in the original `rlist` object now corresponds to elements 3 and 4 of the `rlist2` object. Again, we call `rearDataFrameList` and `ggRearrange` to organize and then plot the supporting reads.

```
df2 <- rearDataFrameList(rlist2[3:4], build="hg19")
ggRearrange(df)
```


5'−noncoding1
3'−noncoding2
5'−noncoding2
3'−noncoding1
read pair index
Mb
read_type
R1−+
R2−+
R1+−
R2+−
splitread