Overview of svalignments package

Robert Scharpf

August 24, 2017

Introduction

The main use-cases of the `svalignments` package are (1) to identify rearranged read pairs that support structural variants, (2) assess BLAT realignment of selected reads for consistency with putative rearrangements and the whole-genome aligner. Most use-cases require a BAM file. For the purpose of illustrating examples in this vignette, we use BAM files in the `svbams` package.

Extract rearranged read pairs from a BAM file for a region of interest

This section describes how to extract rearranged read pairs from a BAM file for a specific genomic region of interest (e.g., a possible deletion). In the following code chunk, we load the `svbams` package which contains a small bam files for a region of chr15.

```
library(GenomicAlignments)
library(Rsamtools)
library(svbams)
library(svalignments)
path <- system.file("extdata", package="svbams")
```

First, we define the region of interest, loading the `TxDb.Hsapiens.UCSC.hg19.refGene` package to extract the sequence length for chromosome 15.

```
library(TxDb.Hsapiens.UCSC.hg19.refGene)
region <- GRanges("chr15", IRanges(63201003, 63209243))
si <- seqinfo(TxDb.Hsapiens.UCSC.hg19.refGene)
seqinfo(region) <- si["chr15", ]
```

To be sure this region is big enough to capture any interesting nearby features, we expand the region of interest to include 5kb on either side of the candidate sequence junctions.

```
region.expand <- GRanges("chr15", IRanges(start(region)-5e3, end(region)+5e3))
seqinfo(region.expand) <- seqinfo(region)
```

Next, we create a BamViews object that will contain metadata on the bamfiles.

```
bampath <- list.files(path, pattern="cgov44t_revised.bam$", full.names=TRUE)
bview <- BamViews(bamPaths=bampath)
```

The following wrapper for `ScanBamParams` provides a default instance of the class with parameters for extracting improperly paired reads. Note that if `which` is missing for `improperAlignmentParams`, all improperly paired reads will be extracted from the BAM file. For large BAM files, this can be very slow.

```
iparams <- improperAlignmentParams(which=region.expand, mapqFilter=30)
pparams <- properAlignmentParams(which=region.expand, mapqFilter=30)
```

Next, we extract a `GAlignmentsPairs` object encapsulating all of the improperly and properly paired reads in this region.

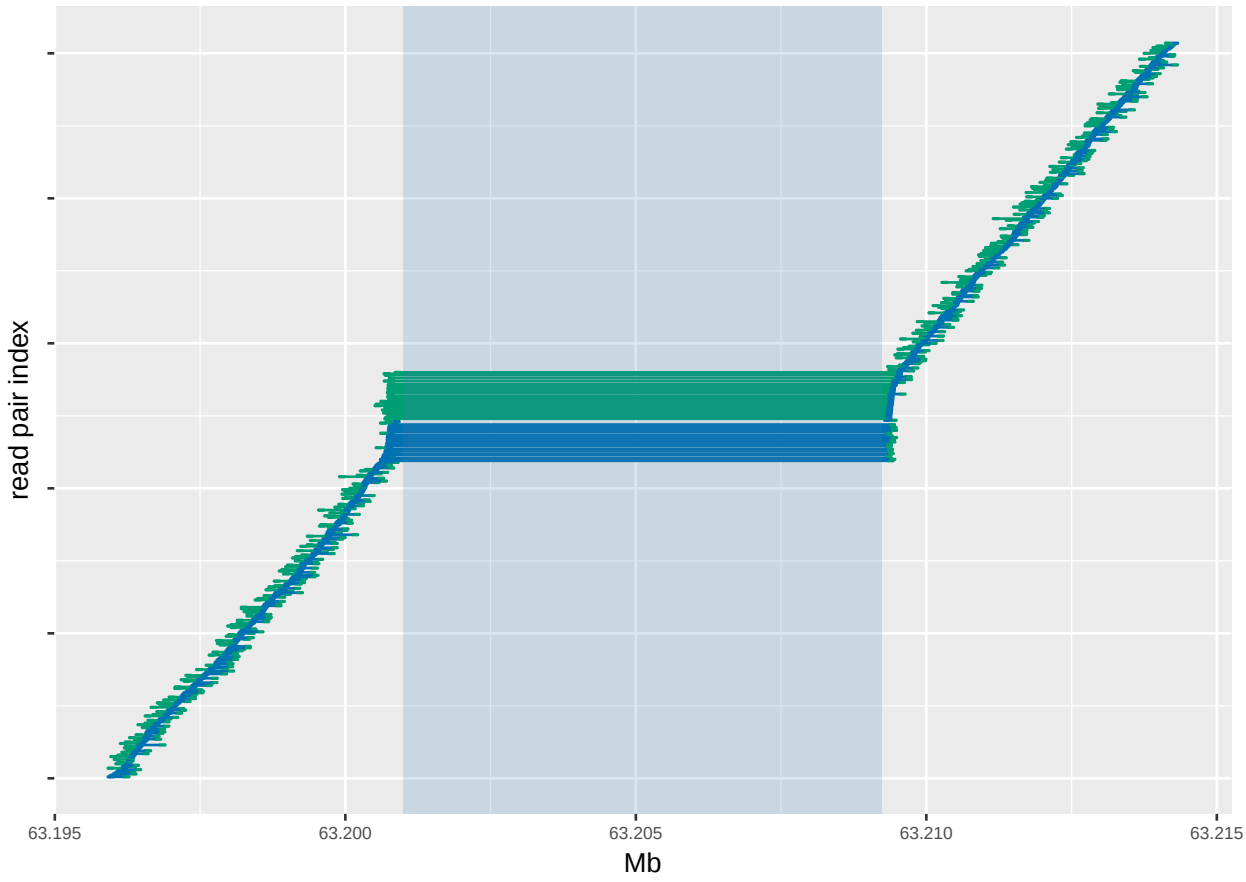
```
irp <- getImproperAlignmentPairs(bampath,
                                 iparams,
                                 build="hg19")
prp <- getProperAlignmentPairs(bampath,
                               pparams,
                               build="hg19")
```

For visualization, we melt the `GAlignmentPairs` to `GRanges` and coerce to a data.frame. Depending on the size of the region, plotting all properly paired reads may not be desirable.

```
igr <- ga2gr(irp, is.improper=TRUE)
pgr <- ga2gr(prp, is.improper=FALSE)
pgr <- thinProperPairs(pgr, 10)
gr <- sortByRead1(c(igr, pgr))
df <- as(gr, "data.frame")
```

```
library(ggplot2)
library(scales)
colors <- c("#0072B2", "#009E73")
df$tagid <- as.numeric(df$tagid)
ggplot(df, aes(ymin=tagid-0.2, ymax=tagid+0.2,
               xmin=start/1e6, xmax=end/1e6, color=read,
               fill=read, group=tagid)) +
  geom_rect() +
  xlim(c(min(df$start), max(df$end))/1e6) +
  geom_line(aes(x=start/1e6, y=tagid)) +
  ylab("read pair index") +
  scale_x_continuous(breaks=pretty_breaks(5)) +
  geom_rect(data=as.data.frame(region),
            aes(xmin=start/1e6, xmax=end/1e6, ymin=-Inf, ymax=+Inf),
            fill="steelblue", color="transparent", alpha=0.2,
            inherit.aes=FALSE) +
  scale_color_manual(values=colors) +
  scale_fill_manual(values=colors) +
  xlab("Mb") +
  theme(axis.text.x=element_text(size=7),
        axis.text.y=element_blank()) +
  guides(fill=FALSE, color=FALSE)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



Extract read pairs from BAM file with exogenous barcodes

DEPRECATE: duplicates can be marked in a way that is barcode-aware. Downstream processing would be identical.

```
targeted.bam <- file.path(path, "PGDX5881P_PS_Seq_novo.sorted_nofa.bam")
library(dplyr)
### Read in GAlignmentPairs
### Not getting rid of duplicates yet
### Not sure if BC tag is unique to pgdx
gr <- GRanges("chr8", IRanges(5000, 100000))
seqinfo(gr) <- Seqinfo(seqnames="chr8", seqlengths=146364022)
param <- ScanBamParam(flag=scanBamFlag(isSecondaryAlignment=FALSE,
                                       isUnmappedQuery=FALSE),
                      which=gr,
                      what="mapq",
                      tag="BC")
galp <- readGAlignmentPairs(targeted.bam, param=param)
read1mapq <- mcols(GenomicAlignments::first(galp))$mapq
read2mapq <- mcols(GenomicAlignments::second(galp))$mapq
galp <- galp[read1mapq >= 30 & read2mapq >= 30]
```

Next, we extract the barcode from the GAlignmentPairs object and convert the paired reads to DNA fragments.

```
### Get barcode
barcode <- mcols(GenomicAlignments::first(galp))$BC
## Only care about this part of the barcode since first 8 BP are same within a
## sample
mcols(galp)$barcode <- substr(barcode, 9, 16)
fragments <- granges(galp, on.discordant.seqnames = "drop",
                     use.mcols=TRUE)
```

Finally, we identify the set of unique fragments, coercing the `GRanges` object to a `data.frame` and grouping the fragments by chromosome, start, end, and barcode. We do this because using the `unique` function on our fragments `GRanges` object will not take into account the barcode.

```
### Now to data.table
fragments.df <- data.frame(chr = as.character(seqnames(fragments)),
                           start = start(fragments),
                           end = end(fragments),
                           barcode = fragments$barcode)
which.isduplicated <- duplicated(fragments.df)
fragments <- fragments[!which.isduplicated]
```

Write selected reads to fasta

TODO