MCB536 Lecture 16 (Part 2): Sequence Data Analysis in R

Gavin Ha

11/24/2020

Loading and querying a BAM file using Rsamtools

The BAM file is the primary input for Rsamtools. There are two initial steps:

a. Define the genomic coordinates and components to query (`ScanBamParam`)
b. Scan the BAM file (`scanBam`)

For this tutorial, we will be using the same data and example from Lecture 15: Slides 12-16. The BAM file can be downloaded at https://www.dropbox.com/home/File%20requests/TFCB_tutorials

For more information, refer to https://bioconductor.org/packages/release/bioc/vignettes/Rsamtools/inst/doc/Rsamtools-Overview.pdf

0 Install and load the `Rsamtools` Bioconductor package

Bioconductor package providing functions to interface with aligned BAM files. https://bioconductor.org/packages/release/bioc/html/Rsamtools.html

```
#BiocManager::install("Rsamtools")
library(Rsamtools)
```

1. Setup parameters for scanning BAM file

a. Specify the genomic location of interest to query in the BAM file.

This will make use of two of the packages (`GRanges`) that we will describe in more detail later.

```
whichRanges <- GRanges(seqnames = "17",
                       IRanges(start = 37844393, end = 37844393))
```

b. Specify which fields to return in the query.

To find out the default fields to return, use `scanBamWhat()`

```
whatFields <- scanBamWhat()
```

c. Specify the filters to use to include or exclude reads.

This is an essential concept in analyzing sequence data. First, specify the status of the reads based on the `FLAG` (recall Lecture 15: Slide 22). For more details, use `?scanBamFlag`

```
flag <- scanBamFlag(isDuplicate = FALSE)  # exclude PCR duplicate reads
```

Next, specify additional filters to use including `mapqFilter`, `tagFilter`. These are included in the final `scanBamParam` object instantiation, along with all the previous arguments.

```
param <- ScanBamParam(flag = flag, which = whichRanges, what = whatFields,
             mapqFilter = 30, tag = c("RG"))
param
```

```
## class: ScanBamParam
## bamFlag (NA unless specified): isDuplicate=FALSE
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag: RG
## bamTagFilter:
## bamWhich: 1 ranges
## bamWhat: qname, flag, rname, strand, pos, qwidth, mapq, cigar, mrnm,
##   mpos, isize, seq, qual, groupid, mate_status
## bamMapqFilter: 30
```

2. Query the BAM file

Using the params we just defined, we will query the BAM file `BRCA_IDC_cfDNA.bam`.

```
bamFile <- "BRCA_IDC_cfDNA.bam"
bam <- scanBam(bamFile, param = param)
```

This returns a `list` object with each element representing a read. For each element/read, there is another `list` with the fields in the BAM file we requested with `scanBamWhat()`. Here is a breakdown of what is in the first read. Refer to Lecture 15: Slides 22.

```
bam[[1]]$qname # reqd query name
```

```
## [1] "41976152"
```

```
bam[[1]]$flag  # bitwise flag describing the read alignment
```

```
## [1] 163
```

```
bam[[1]]$rname # reference sequence name
```

```
## [1] 17
## 86 Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y ... hs37d5
```

```
bam[[1]]$pos   # position of aligned read (leftmost coordinate)
```

```
## [1] 37844359
```

```
bam[[1]]$mapq  # mapping quality of the read alignment
```

```
## [1] 60
```

```
bam[[1]]$cigar # CIGAR string
```

```
## [1] "39M"
```

```
bam[[1]]$mrnm  # mate read's reference sequence name
```

```
## [1] 17
```

```
## 86 Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y ... hs37d5
```

```
bam[[1]]$mpos  # mate read's aligned position
```

```
## [1] 37844477
```

```
bam[[1]]$isize # insert size or templent length; aka fragment size
```

```
## [1] 157
```

```
as.character(bam[[1]]$seq)   # sequence of mapped reads on forward strand
```

```
## [1] "ACTCTCCGCTGAAGTCCACACAGTTTAAATTAAAGTTCC"
```

```
as.character(bam[[1]]$qual)  # base qualities of the sequence aligment
```

```
## [1] ".AAAAFFFFFFFFFFFF)FAFFFFFFFFFFFFFFFFFF"
```

```
bam[[1]]$tag   # value for the tag we specified
```

```
## $RG
## [1] "P12.17.7_Breast"
```

Exercise 2: Extract sequence data information

a. Create a range for `11:69462758-69462758`.

```
# GRanges()
```

b. Specify the BAM query parameters.

```
# scanBamWhat()

# scanBamFlag()

# ScanBamParam()
```

c. What is the sequence of the read at `11:69462758-69462758`?

```
# scanBam()
```

2. Compute "Pile-Up" Statistics

The pileup is a term referring to counting the alleles from all the reads at a given genomic locus. It is the data that many variant and mutation calling algorithms use to determine variant status and allelic fractions.

There are 3 steps:

a. Define the genomic coordinates and read components to query (`ScanBamParam`) - same as before
b. Define the pileup-specific parameters, such as filters (`PileupParam`)
c. Run the `pileup` command

a. Set up the pileup parameters: `PileupParam`.

The `PileupParam()` function will allow for specifying criteria such as minimum read depth, https://www.rdocumentation.org/packages/Rsamtools/versions/1.24.0/topics/pileup

```
pu.param <- PileupParam() # default settings
```

b. Set up scanBam parameters: `ScanBamParam`.

Let's try generating the pileup for `17:37883255-37883260`.

```
whichRanges2 <- GRanges(seqnames = "17",
                        IRanges(start = 37883255, end = 37883260))
param <- ScanBamParam(flag = flag, which = whichRanges2,
                      what = whatFields, mapqFilter = 30)
```

c. Generate the pipeline at `17:37883255-37883260`.

The `pileup` command outputs a `data.frame` object containing the counts for each allele at every base specified in `param`.

```
pu <- pileup(file = bamFile, scanBamParam = param, pileupParam = pu.param)
```