An Introduction to *Rsamtools*

Martin Morgan

Modified: 18 March, 2010. Compiled: January 12, 2011

```
> library(Rsamtools)
```

1 Introduction

The *Rsamtools* package provides an interface to `BAM` files. `BAM` files are produced by samtools and other software, and represent a flexible format for storing 'short' reads aligned to reference genomes. `BAM` files typically contain sequence and base qualities, and alignment coordinates and quality measures. `BAM` files are appealing for several reasons. The format is flexible enough to represent reads generated and aligned using diverse technologies. The files are binary so that file access is relatively efficient. `BAM` files can be indexed, allowing ready access to localized chromosomal regions. `BAM` files can be accessed remotely, provided the remote hosting site supports such access and a local index is available. This means that specific regions of remote files can be accessed without retrieving the entire (large!) file. A full description is available in the `BAM` format specification (`http://samtools.sourceforge.net/SAM1.pdf`)

The main purpose of the *Rsamtools* package is to import `BAM` files into R. *Rsamtools* also provides some facility for file access such as record counting, index file creation, and filtering to create new files containing subsets of the original. An important use case for *Rsamtools* is as a starting point for creating R objects suitable for a diversity of work flows, e.g., *AlignedRead* objects in the *ShortRead* package (for quality assessment and read manipulation), or *GappedAlignments* objects in *GenomicRanges* package (for RNA-seq and other applications). Those desiring more functionality are encouraged to explore samtools and related software efforts.

2 Input

The essential capability provided by *Rsamtools* is `BAM` input. This is accomplished with the `scanBam` function. `scanBam` takes as input the name of the `BAM` file to be parsed. In addition, the `param` argument determines which genomic coordinates of the `BAM` file, and what components of each record, will be input. `param` is an instance of the *ScanBamParam* class. To create a `param` object, call `ScanBamParam`. Here we create a `param` object to extract reads aligned to three

distinct ranges (one on `seq1`, two on `seq2`). From each of read in those ranges, we specify that we would like to extract the reference name (`rname`, e.g., `seq1`), strand, alignment position, query (i.e., read) width, and query sequence:

```
> which <- RangesList(seq1 = IRanges(1000, 2000),
+     seq2 = IRanges(c(100, 1000), c(1000, 2000)))
> what <- c("rname", "strand", "pos", "qwidth",
+     "seq")
> param <- ScanBamParam(which = which, what = what)
```

Additional information can be found on the help page for `ScanBamParam`. Reading the relevant records from the `BAM` file is accomplished with

```
> bamFile <-
+     system.file("extdata", "ex1.bam", package="Rsamtools")
> bam <- scanBam(bamFile, param=param)
```

Like `scan`, `scanBam` returns a `list` of values. Each element of the list corresponds to a range specified by the `which` argument to `ScanBamParam`.

```
> class(bam)

[1] "list"

> names(bam)

[1] "seq1:1000-2000" "seq2:100-1000"  "seq2:1000-2000"
```

Each element is itself a list, containing the elements specified by the `what` and `tag` arguments to `ScanBamParam`.

```
> class(bam[[1]])

[1] "list"

> names(bam[[1]])

[1] "rname"  "strand" "pos"    "qwidth" "seq"
```

The elements are either basic R or *IRanges* data types

```
> sapply(bam[[1]], class)

         rname         strand            pos         qwidth
      "factor"       "factor"      "integer"      "integer"
           seq
"DNAStringSet"
```

A paradigm for collapsing the list-of-lists into a single list is

```
> lst <- lapply(names(bam[[1]]), function(elt) {
+     do.call(c, unname(lapply(bam, "[[", elt)))
+ })
> names(lst) <- names(bam[[1]])
```

This might be further transformed, e.g., to a *DataFrame*, with

```
> head(do.call("DataFrame", lst))
```

```
DataFrame with 6 rows and 5 columns
      rname    strand       pos    qwidth
  <integer> <integer> <integer> <integer>
1         1         1       970        35
2         1         1       971        35
3         1         1       972        35
4         1         1       973        35
5         1         1       974        35
6         1         2       975        35
                                 seq
                         <DNAStringSet>
1 TATTAGGAAATGCTTTACTGTCATAACTATGAAGA
2 ATTAGGAAATGCTTTACTGTCATAACTATGAAGAG
3 TTAGGAAATGCTTTACTGTCATAACTATGAAGAGA
4 TAGGAAATGCTTTACTGTCATAACTATGAAGAGAC
5 AGGAAATGCTTTACTGTCATAACTATGAAGAGACT
6 GGAAATGCTTTACTGTCATAACTATGAAGAGACTA
```

The `BAM` file in the previous example includes an index, represented by a separate file with extension `.bai`:

```
> list.files(dirname(bamFile), pattern = "ex1.bam(.bai)?")
```

```
[1] "ex1.bam"     "ex1.bam.bai"
```

Indexing provides two significant benefits. First, an index allows a `BAM` file to be efficiently accessed by range. A corollary is that providing a `which` argument to `ScanBamParam` requires an index. Second, coordinates for extracting information from a `BAM` file can be derived from the index, so a portion of a remote `BAM` file can be retrieved with local access only to the index. For instance, provided an index file exists on the local computer, it is possible to retrieve a small portion of a `BAM` file residing on the 1000 genomes HTTP server. The url `ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/pilot_data/data/NA19240/alignment/NA19240.chrom6.SLX.maq.SRP000032.2009_07.bam` points to the `BAM` file corresponding to individual NA19240 chromosome 6 Solexa (Illumina) sequences aligned using MAQ. The remote file is very large (about 10 GB), but the corresponding index file is small (about 500 KB). With `na19240url` set to the above address, the following retrieves just those reads in the specified range

```
> which <- RangesList(`6` = IRanges(100000L, 110000L))
> param <- ScanBamParam(which = which)
> na19240bam <- scanBam(na19240url, param = param)
```

Invoking `scanBam` without an index file, as above, first retrieves the index file from the remote location, and then queries the remote file using the index; for repeated queries, it is more efficient to retrieve the index file first (e.g., with `download.file`) and then use the local index as an argument to `scanBam`. Many `BAM` files were created in a way that causes `scanBam` to report that the "EOF marker is absent"; this message can safely be ignored.

`BAM` files may be read by functions in packages other than *Rsamtools*. In *ShortRead*, `readAligned` invoked with `type="bam"` will read `BAM` files in to an *AlignedRead* object. This function takes a `param` argument, just as `scanBam`, so the user can control which portions of the file are input. Similar input facilities are available through the `readGappedAlignment` function in *GenomicRanges*.

Additional ways of interacting with `BAM` files include `scanBamHeader` (to extract header information) and `countBam` (to count records matching `param`). `filterBam` filters reads from the source file according to the criteria of the *ScanBamParam* parameter, writing reads passing the filter to a new file. The function `sortBam` sorts a previously unsorted `BAM`, while The function `indexBam` creates an index file from a sorted `BAM` file.

`readPileup` reads a `pileup` file created by samtools, importing SNP, indel, or all variants into a *GRanges* object.

2.1 Large bam files

`BAM` files can be large, containing more information on more genomic regions than are of immediate interest or than can fit in memory. The first strategy for dealing with this is to select, using the `what` and `which` arguments to `ScanBamParam`, just those portions of the `BAM` file that are essential to the current analysis, e.g., specifying `what=c('rname', 'qname', 'pos')` when wishing to calculate coverage of ungapped reads.

When selective input of `BAM` files is still too memory-intensive, the file can be processed in chunks, with each chunk distilled to the derived information of interest. Chromosomes will often be the natural chunk to process. For instance, here we write a summary function that takes a single sequence name (chromosome) as input, reads in specific information from the `BAM` file, and calculates coverage over that sequence.

```
> summaryFunction <- function(seqname, bamFile,
+     ...) {
+     param <- ScanBamParam(what = c("pos", "qwidth"),
+         which = GRanges(seqname, IRanges(1, 1e+07)),
+         flag = scanBamFlag(isUnmappedQuery = FALSE))
+     x <- scanBam(bamFile, ..., param = param)[[1]]
+     coverage(IRanges(x[["pos"]], width = x[["qwidth"]]))
+ }
```

This might be used as follows; it is an ideal candidate for evaluation in parallel, e.g., using the *multicore* package and `srapply` function in *ShortRead*.

```
> seqnames <- paste("seq", 1:2, sep = "")
> cvg <- lapply(seqnames, summaryFunction, bamFile)
> names(cvg) <- seqnames
> cvg

$seq1
'integer' Rle of length 1569 with 1054 runs
  Lengths:  2  2  1  3  4  2  3  4 ...  1  1  1  1  1  1  1
  Values :  1  2  3  4  5  7  8  9 ...  9  7  6  5  3  2  1

$seq2
'integer' Rle of length 1567 with 1092 runs
  Lengths:  1  3  1  1  1  3  1  4 ...  1  1  2  1  4  4  1
  Values :  3  4  5  8 12 14 15 16 ... 10  8  7  6  3  2  1
```

The result of the function (a coverage vector, in this case) will often be much smaller than the input.

3 Views

The functions described in the previous section import data in to R. However, sequence data can be very large, and it does not always make sense to read the data in immediately. Instead, it can be useful to marshal *references* to the data into a container and then act on components of the container. The *BamViews* class provides a mechanism for creating 'views' into a set of `BAM` files. The view itself is light-weight, containing references to the relevant `BAM` files and metadata about the view (e.g., the phenotypic samples corresponding to each `BAM` file).

One way of understanding a *BamViews* instance is as a rectangular data structure. The columns represent `BAM` files (e.g., distinct samples). The rows represent ranges (i.e., genomic coordinates). For instance, a ChIP-seq experiment might identify a number of peaks of high read counts.

3.1 Assembling a *BamViews* instance

To illustrate, suppose we have an interest in caffeine metabolism in humans. The 'rows' contain coordinates of genomic regions associated with genes in a KEGG caffeine metabolism pathway. The 'columns' represent individuals in the 1000 genomes project. The details of creating a *BamViews* object are described in the appendix. Here we retrieve archived versions of the genomic ranges we are interested in (`bamRanges`) and a vector of `BAM` file URLs (`slxMaq09`).

```
> library(GenomicFeatures)
> bamRanges <- local({
```

```
+     fl <- system.file("extdata", "CaffeineTxdb.sqlite",
+         package = "Rsamtools")
+     transcripts(loadFeatures(fl))
+ })
> slxMaq09 <- local({
+     fl <- system.file("extdata", "slxMaq09_urls.txt",
+         package = "Rsamtools")
+     readLines(fl)
+ })
```

We now assemble the *BamViews* instance from these objects; we also provide information to annotated the `BAM` files (with the `bamSamples` function argument, which is a *DataFrame* instance with each row corresponding to a `BAM` file) and the instance as a whole (with `bamExperiment`, a simple named *list* containing information structured as the user sees fit).

```
> bamExperiment <-
+     list(description="Caffeine metabolism views on 1000 genomes samples",
+          created=date())
> bv <- BamViews(slxMaq09, bamRanges=bamRanges,
+                bamExperiment=bamExperiment)
> metadata(bamSamples(bv)) <-
+     list(description="Solexa/MAQ samples, August 2009",
+          created="Thu Mar 25 14:08:42 2010")
```

3.2 Using *BamViews* instances

The *BamViews* object can be queried for its component parts, e.g.,

```
> bamExperiment(bv)
```

```
$description
[1] "Caffeine metabolism views on 1000 genomes samples"

$created
[1] "Wed Jan 12 03:20:09 2011"
```

More usefully, Methods in *Rsamtools* are designed to work with *BamViews* objects, retrieving data from all files in the view. These operations can take substantial time and require reliable network access.

To illustrate, the following code (not evaluated when this vignette was created) downloads the index files associated with the `bv` object

```
> bamIndexDir <- tempfile()
> indexFiles <- paste(bamPaths(bv), "bai", sep = ".")
> dir.create(bamIndexDir)
> idxFiles <- mapply(download.file, indexFiles,
+     file.path(bamIndexDir, basename(indexFiles)),
+     MoreArgs = list(method = "curl"))
```

and then queries the 1000 genomes project for reads overlapping our transcripts; by loading the *multicore* package, we tell *Rsamtools* to use as many cores as are available on our machine (the *multicore* package is not currently – as of 25 March, 2010 – reliable on Windows computers)

```
> library(multicore)
> olaps <- readBamGappedAlignments(bv)
```

The resulting object is about 6 MB in size. To avoid having to download this data each time the vignette is run, we instead load it from disk

```
> load(system.file("extdata", "olaps.Rda", package = "Rsamtools"))
> olaps
```

```
SimpleList of length 24
names(24): NA06986.SLX.maq.SRP000031.2009_08.bam ...
```

```
> head(olaps[[1]])
```

```
GappedAlignments of length 6
    rname strand cigar qwidth    start      end width ngap
[1]    19      +   51M     51 41594322 41594372    51    0
[2]    19      -   51M     51 41594351 41594401    51    0
[3]    19      -   51M     51 41594370 41594420    51    0
[4]    19      +   51M     51 41594371 41594421    51    0
[5]    19      -   51M     51 41594402 41594452    51    0
[6]    19      +   51M     51 41594413 41594463    51    0
```

There are 12904 reads in `NA06986.SLX.maq.SRP000031.2009_08.bam` overlapping at least one of our transcripts. It is easy to explore this object, for instance discovering the range of read widths in each individual.

```
> head(t(sapply(olaps, function(elt) range(qwidth(elt)))))
```

```
                                       [,1] [,2]
NA06986.SLX.maq.SRP000031.2009_08.bam   51   51
NA06994.SLX.maq.SRP000031.2009_08.bam   36   51
NA07051.SLX.maq.SRP000031.2009_08.bam   51   51
NA07346.SLX.maq.SRP000031.2009_08.bam   48   76
NA07347.SLX.maq.SRP000031.2009_08.bam   51   51
NA10847.SLX.maq.SRP000031.2009_08.bam   36   51
```

Suppose we were particularly interested in the first transcript, which has a transcript id ENST00000343932. Here we extract reads overlapping this transcript from each of our samples. As a consequence of the protocol used, reads aligning to either strand could be derived from this transcript. For this reason, we set the strand of our range of interest to `*`. We use the `endoapply` function, which is like `lapply` but returns an object of the same class (in this case, SimpleList) as its first argument.

```
> rng <- bamRanges(bv)[1]
> strand(rng) <- "*"
> olap1 <- endoapply(olaps, subsetByOverlaps, rng)
> olap1

SimpleList of length 24
names(24): NA06986.SLX.maq.SRP000031.2009_08.bam ...

> head(olap1[[24]])

GappedAlignments of length 6
    rname strand cigar qwidth    start      end width ngap
[1]    15      +   36M     36 75041174 75041209    36    0
[2]    15      -   36M     36 75041180 75041215    36    0
[3]    15      +   36M     36 75041186 75041221    36    0
[4]    15      -   36M     36 75041206 75041241    36    0
[5]    15      +   36M     36 75041208 75041243    36    0
[6]    15      +   36M     36 75041211 75041246    36    0
```

To carry the example a little further, we calculate coverage of each sample:

```
> minw <- min(sapply(olap1, function(elt) min(start(elt))))
> maxw <- max(sapply(olap1, function(elt) max(end(elt))))
> cvg <- endoapply(olap1, coverage,
+                  shift=-start(ranges(bamRanges[1])),
+                  width=width(ranges(bamRanges[1])))
> cvg

SimpleList of length 24
names(24): NA06986.SLX.maq.SRP000031.2009_08.bam ...

> cvg[[1]]

SimpleRleList of length 25
$`1`
'integer' Rle of length 7758 with 1 run
  Lengths: 7758
  Values :    0

$`2`
'integer' Rle of length 7758 with 1 run
  Lengths: 7758
  Values :    0

$`3`
'integer' Rle of length 7758 with 1 run
  Lengths: 7758
  Values :    0
```

```
$`4`
'integer' Rle of length 7758 with 1 run
  Lengths: 7758
  Values :    0

$`5`
'integer' Rle of length 7758 with 1 run
  Lengths: 7758
  Values :    0

...
<20 more elements>
```

Since the example includes a single region of uniform width across all samples, we can easily create a coverage matrix with rows representing nucleotide and columns sample and, e.g., document variability between samples and nucleotides

```
> m <- matrix(unlist(lapply(cvg, lapply, as.vector)),
+     ncol = length(cvg))
> summary(rowSums(m))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.000   3.254   0.000 127.000

> summary(colSums(m))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  13220   15260   23400   26300   35590   56840
```

4 Directions

This vignette has summarized facilities in the *Rsamtools* package. Important additional packages include *GenomeRanges* (for representing and manipulating gapped alignments), *ShortRead* for I/O and quality assessment of ungapped short read alignments, *Biostrings* and *BSgenome* for DNA sequence and whole-genome manipulation, *IRanges* for range-based manipulation, and *rtracklayer* for I/O related to the UCSC genome browser. Users might also find the interface to the integrative genome browser (IGV) in *SRAdb* useful for visualizing `BAM` files.

```
> packageDescription("Rsamtools")

Package: Rsamtools
Type: Package
Title: Import aligned BAM file format sequences into
      R / Bioconductor
```

```
Version: 1.2.3
Author: Martin Morgan, Herv\'e Pag\`es
Maintainer: Biocore Team c/o BioC user list
      <bioconductor@stat.math.ethz.ch>
Description: This package provides an interface to
      the 'samtools' utilities for manipulating SAM
      (Sequence Alignment / Map) format files.
      Facilities currently available include flexible
      file input.
URL:
      http://bioconductor.org/packages/release/bioc/html/Rsamtools.html
License: Artistic-2.0
LazyLoad: yes
Depends: methods, IRanges (>= 1.7.23), GenomicRanges
      (>= 0.1.0), Biostrings (>= 2.15.0)
Imports: methods, utils, IRanges, GenomicRanges,
      Biostrings
Suggests: ShortRead, GenomicFeatures, RUnit, KEGG.db
LinkingTo: Biostrings, IRanges
biocViews: DataImport, Sequencing,
      HighThroughputSequencing
Built: R 2.12.1; x86_64-unknown-linux-gnu; 2011-01-12
      11:19:50 UTC; unix

-- File: /tmp/Rtmp5oQGrr/Rinst3fbe66d3/Rsamtools/Meta/package.rds

> sessionInfo()

R version 2.12.1 (2010-12-16)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=C              LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets
[6] methods   base

other attached packages:
[1] GenomicFeatures_1.2.3 Rsamtools_1.2.3
[3] Biostrings_2.18.2     GenomicRanges_1.2.3
```

```
[5] IRanges_1.8.8

loaded via a namespace (and not attached):
[1] BSgenome_1.18.2    Biobase_2.10.0     DBI_0.2-5
[4] RCurl_1.5-0        RSQLite_0.9-4      XML_3.2-0
[7] biomaRt_2.6.0      rtracklayer_1.10.6 tools_2.12.1
```

A Assembling a *BamViews* instance

A.1 Genomic ranges of interest

We identify possible genes that KEGG associates with caffeine metabolism:

```
> library(KEGG.db)
> kid <- revmap(KEGGPATHID2NAME)[["Caffeine metabolism"]]
> egid <- KEGGPATHID2EXTID[[sprintf("hsa%s", kid)]]
```

Then we use the *biomaRt* package to translate Entrez identifiers to Ensembl transcript ids, and use these to create a small data base (using the *GenomicFeatures* packages) of features we are interested in

```
> library(biomaRt)
> mart <- useMart("ensembl", "hsapiens_gene_ensembl")
> ensid <- getBM(c("ensembl_transcript_id"), filters = "entrezgene",
+     values = egid, mart = mart)[[1]]
> library(GenomicFeatures)
> txdb <- makeTranscriptDbFromBiomart(transcript_ids = ensid)
```

Transcript genomic locations are contained in `txdb`; we will use those as the ranges for the *BamViews*

```
> bamRanges <- transcripts(txdb)
```

A.2 BAM files

Note: The following operations were performed at the time the vignette was written; location of on-line resources, in particular the organization of the 1000 genomes `BAM` files, may have changed.

We are interested in collecting the URLs of a number of `BAM` files from the 1000 genomes project. Our first goal is to identify files that might make for an interesting comparison. First, let's visit the 1000 genomes FTP site and discover available files. We'll use the *RCurl* package to retrieve the names of all files in an appropriate directory

```
> library(RCurl)
> ftpBase <-
+     "ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/pilot_data/data/"
> indivDirs <-
```

```
+     strsplit(getURL(ftpBase, ftplistonly=TRUE), "\n")[[1]]
> alnDirs <-
+     paste(ftpBase, indivDirs, "/alignment/", sep="")
> urls0 <-
+     strsplit(getURL(alnDirs, dirlistonly=TRUE), "\n")
```

From these, we exclude directories without any files in them, select only the BAM index (extension .bai) files, and choose those files that exactly six '.' characters in their name.

```
> urls <- urls[sapply(urls0, length) != 0]
> fls0 <- unlist(unname(urls0))
> fls1 <- fls0[grepl("bai$", fls0)]
> fls <- fls1[sapply(strsplit(fls1, "\\."), length) ==
+     7]
```

After a little exploration, we focus on those files obtained form Solexa sequencing, aligned using MAQ, and archived in August, 2009; we remove the .bai extension, so that the URL refers to the underlying file rather than index. There are 24 files.

```
> urls1 <- Filter(function(x) length(x) != 0, lapply(urls,
+     function(x) x[grepl("SLX.maq.*2009_08.*bai$",
+         x)]))
> slxMaq09.bai <- mapply(paste, names(urls1), urls1,
+     sep = "", USE.NAMES = FALSE)
> slxMaq09 <- sub(".bai$", "", slxMaq09.bai)
```

As a final step to prepare for using a *BamViews* file, we create local copies of the *index* files. The index files are relatively small (about 190 Mb total).

```
> bamIndexDir <- tempfile()
> dir.create(bamIndexDir)
> idxFiles <- mapply(download.file, slxMaq09.bai,
+     file.path(bamIndexDir, basename(slxMaq09.bai)),
+     MoreArgs = list(method = "curl"))
```