Managing and analyzing multiple NGS samples with Bioconductor bamViews objects: application to RNA-seq

VJ Carey

May 2, 2024

Contents

1 Introduction 2

2 Basic design 2

3 Illustration 2

4 Comparative counts in a set of regions of interest 5
   4.1 Counts in a regular partition 5
   4.2 Counts in annotated intervals: genes 7

5 Larger scale sanity check 8

6 Statistical analyses of differential expression 9
   6.1 Using edgeR 9

7 Summary 11

8 Session data 12
1 Introduction

We consider a lightweight approach to Bioconductor-based management and interrogation of multiple samples to which NGS methods have been applied.

The basic data store is the binary SAM (BAM) format [Li et al. 2009]. This format is widely used in the 1000 genomes project, and transformations between SAM/BAM and output formats of various popular alignment programs are well-established. Bioconductor’s Rsamtools package allows direct use of important SAM data interrogation facilities from R.

2 Basic design

A collection of NGS samples is represented through the associated set of BAM files and BAI index files. These can be stored in the inst/bam folder of an R package to facilitate documented programmatic access through R file navigation facilities, or the BAM/BAI files can be accessed through arbitrary path or URL references.

The bamViews class is defined to allow reliable fine-grained access to the NGS data along with relevant metadata. A bamViews instance contains access path information for a set of related BAM/BAI files, along with sample metadata and an optional specification of genomic ranges of interest.

A key design aspect of the bamViews class is preservation of semantics of the X[G, S] idiom familiar from ExpressionSet objects for management of multiple microarrays. With ExpressionSet instances, G is a predicate specifying selection of microarray probes of interest. With bamViews instances, G is a predicate specifying selection of genomic features of interest. At present, for bamViews, selection using G involves ranges of genomic coordinates.

3 Illustration

Data from four samples from a yeast RNA-seq experiment (two wild type, two ‘RLP’ mutants) are organized in the leeBamViews package. The data are collected to allow regeneration of aspects of Figure 8 of Lee et al. [2008]. We obtained all reads between bases 800000 and 900000 of yeast chromosome XII.

We have not yet addressed durable serialization of manager objects, so the bamViews instance is created on the fly.

> suppressPackageStartupMessages({
+ library(leeBamViews)  # bam files stored in package
+ library(S4Vectors)
+ })
> bpaths = dir(system.file("bam", package="leeBamViews"), full=TRUE, patt="bam$")
> #
> # extract genotype and lane information from filenames
> #
> gt = gsub(".*/", "", bpaths)
> gt = gsub("_.*", "", gt)
> lane = gsub(".*(.:.)$", "\1", gt)
> geno = gsub(".$", "", gt)
> #
> # format the sample-level information appropriately
> #
> pd = DataFrame(geno=geno, lane=lane, row.names=paste(geno, lane, sep="."))
> prd = new("DFrame") # protocol data could go here
> #
> # create the views object, adding some arbitrary experiment-level information
> #
> bs1 = BamViews(bamPaths=bpaths, bamSamples=pd,
+     bamExperiment=list(annotation="org.Sc.sgd.db"))
> bs1

BamViews dim: 0 ranges x 8 samples
names: isowt.5 isowt.6 ... xrn.1 xrn.2
detail: use bamPaths(), bamSamples(), bamRanges(), ...

> #
> # get some sample-level data
> #
> bamSamples(bs1)$geno

[1] "isowt" "isowt" "rlp" "rlp" "ssr" "ssr" "xrn" "xrn"

We would like to operate on specific regions of chr XIII for all samples. Note that the aligner in use (bowtie) employed “Scchr13” to refer to this chromosome. We add a GRanges instance to the view to identify the region of interest.

> START=c(861250, 863000)
> END=c(862750, 864000)
> exc = GRanges(seqnames="Scchr13", IRanges(start=START, end=END), strand="+")
> bamRanges(bs1) = exc
> bs1

BamViews dim: 2 ranges x 8 samples
names: isowt.5 isowt.6 ... xrn.1 xrn.2
detail: use bamPaths(), bamSamples(), bamRanges(), ...

3
A common operation will be to extract coverage information. We use a transforming method, readGAlignments, from the GenomicAlignments package to extract reads and metadata for each region and each sample.

```r
> library(GenomicAlignments)
> covex =
+   RleList(lapply(bamPaths(bs1), function(x)
+     coverage(readGAlignments(x))[["Scchr13"]]))
> names(covex) = gsub(".bam\$", ",", basename(bamPaths(bs1)))
> head(covex, 3)

RleList of length 3

$\text{isowt5}\_13e
integer-Rle of length 924429 with 21819 runs
Lengths: 799974 2 2 1 6 ... 10 7 1 24399
Values : 0 1 2 3 4 ... 4 3 2 0

$\text{isowt6}\_13e
integer-Rle of length 924429 with 21799 runs
Lengths: 799976 2 3 14 13 ... 17 1 4 24394
Values : 0 1 2 3 4 ... 3 2 1 0

$\text{rlp5}\_13e
integer-Rle of length 924429 with 23037 runs
Lengths: 799974 2 6 25 3 ... 4 2 30 24397
Values : 0 1 2 3 4 ... 3 2 1 0

Let's visualize what we have so far. We use GenomeGraphs and add some supporting software. Get a copy from an archive if you want to run this code.

```r
> library(GenomeGraphs)
> cov2baseTrack = function(rle, start, end,
+   dp = DisplayPars(type="l", lwd=0.5, color="black"),
+   countTx=function(x)log10(x+1)) {
+   require(GenomeGraphs)
+   if (!is(rle, "Rle")) stop("requires instance of Rle")
+   dat = runValue(rle)
+   loc = cumsum(runLength(rle))
+   ok = which(loc >= start & loc <= end)
+   makeBaseTrack(base = loc[ok], value=countTx(dat[ok]),
+     dp=dp)
+ }
> trs = lapply(covex, function(x) cov2baseTrack(x, START[1], END[1],
+ countTx = function(x)pmin(x, 80))
> ac = as.character
> names(trs) = paste(ac(bamSamples(bs1)$geno), ac(bamSamples(bs1)$lane), sep="")
> library(biomaRt)
> mart = useMart("ensembl", "scerevisiae_gene_ensembl")
> gr = makeGeneRegion(START, END, chromosome="XIII",
+ strand="+", biomart=mart, dp=DisplayPars(plotId=TRUE,
+ idRotation=0, idColor="black"))
> trs[[length(trs)+1]] = gr
> trs[[length(trs)+1]] = makeGenomeAxis()

> print( gdPlot( trs, minBase=START[1], maxBase=END[1]) )

We can encapsulate this to something like:

> plotStrains = function(bs, query, start, end, snames, mart, chr, strand="+") {
+ filtbs = bs[query, ]
+ cov = lapply(filtbs, coverage)
+ covtrs = lapply(cov, function(x) cov2baseTrack(x[[1]], start, end,
+ countTx = function(x) pmin(x,80)))
+ names(covtrs) = snames
+ gr = makeGeneRegion(start, end, chromosome=chr,
+ strand=strand, biomart=mart, dp=DisplayPars(plotId=TRUE,
+ idRotation=0, idColor="black"))
+ grm = makeGeneRegion(start, end, chromosome=chr,
+ strand="-", biomart=mart, dp=DisplayPars(plotId=TRUE,
+ idRotation=0, idColor="black"))
+ covtrs[[length(covtrs)+1]] = gr
+ covtrs[[length(covtrs)+1]] = makeGenomeAxis()
+ covtrs[[length(covtrs)+1]] = grm
+ gdPlot( covtrs, minBase=start, maxBase=end )
+ }

4 Comparative counts in a set of regions of interest

4.1 Counts in a regular partition

The supplementary information for the Lee paper includes data on unannotated transcribed regions reported in other studies. We consider the study of David et al., confining attention to chromosome XIII. If you wanted to study their intervals you could use code like:
but our object \( \text{bs1} \) is too restricted in its coverage. Instead, we illustrate with a small set of subintervals of the basic interval in use:

\[
\begin{align*}
\text{myrn} &= \text{GRanges(seqnames="Scchr13",} \\
&\quad \text{IRanges(start=seq(861250, 862750, 100), width=100), strand="+"}) \\
\text{elementMetadata(myrn)$name = paste("til", 1:length(myrn), sep=".")} \\
\text{bamRanges(\text{bs1}) = myrn} \\
\text{tabulateReads(\text{bs1}, "}\text{+"})
\end{align*}
\]

\[
\begin{array}{cccccccccccc}
til.1 & til.2 & til.3 & til.4 & til.5 & til.6 & til.7 & til.8 & til.9 & til.10 \\
\hline
\text{start} & 861250 & 861350 & 861450 & 861550 & 861650 & 861750 & 861850 & 861950 & 862050 & 862150 \[0.5ex]
\text{end} & 861349 & 861449 & 861549 & 861649 & 861749 & 861849 & 861949 & 862049 & 862149 & 862249 \\
\text{isowt.5} & 1 & 1 & 3 & 6 & 2 & 7 & 299 & 605 & 408 & 380 \\
\text{isowt.6} & 2 & 6 & 9 & 12 & 7 & 4 & 306 & 666 & 458 & 382 \\
\text{rlp.5} & 1 & 5 & 65 & 53 & 36 & 11 & 158 & 247 & 186 & 145 \\
\text{rlp.6} & 3 & 2 & 47 & 48 & 37 & 16 & 123 & 238 & 163 & 159 \\
\text{ssr.1} & 2 & 6 & 35 & 27 & 21 & 8 & 423 & 700 & 541 & 496 \\
\text{ssr.2} & 2 & 6 & 43 & 37 & 26 & 13 & 443 & 839 & 616 & 509 \\
\text{xrn.1} & 7 & 8 & 75 & 78 & 78 & 24 & 5 & 180 & 446 & 357 & 288 \\
\text{xrn.2} & 4 & 9 & 96 & 110 & 31 & 8 & 225 & 611 & 465 & 356 \\
\end{array}
\]
4.2 Counts in annotated intervals: genes

We can use Bioconductor annotation resources to acquire boundaries of yeast genes on our subregion of chromosome 13.

In the following chunk we generate annotated ranges of genes on the Watson strand.

```r
> library(org.Sc.sgd.db)
> library(IRanges)
> c13g = get("13", revmap(org.Sc.sgdCHR)) # all genes on chr13
> c13loc = unlist(mget(c13g, org.Sc.sgdCHRLOC)) # their 'start' addresses
> c13locend = unlist(mget(c13g, org.Sc.sgdCHRLOCEND))
> c13locp = c13loc[c13loc>0] # confine attention to + strand
> c13locendp = c13locend[c13locend>0]
> ok = !is.na(c13locp) & !is.na(c13locendp)
> c13pr = GRanges(seqnames="Scchr13", IRanges(c13locp[ok], c13locendp[ok]),
+ strand="+") # store and clean up names
> elementMetadata(c13pr)$name = gsub(".13$", "", names(c13locp[ok]))
> c13pr
```

GRanges object with 296 ranges and 1 metadata column:

```
  seqnames    ranges   strand |    name
         <Rle>   <IRanges> <Rle> | <character>
[1] Scchr13 267174-267800   + |  YML001W
[3] Scchr13 263483-264355   + |  YML003W
[4] Scchr13 260221-261609   + |  YML005W
[5] Scchr13 253848-255800   + |  YML007W
... ... ... ... ... ...
[292] Scchr13 611314-611508   + |  YMR175W-A
[293] Scchr13 769283-769426   + |  YMR247W-A
[294] Scchr13 923541-924431   + |   TEL13R
[295] Scchr13 91970-92027     + |   ZOD1
[296] Scchr13 667324-667346   + |     ETC5
```

seqinfo: 1 sequence from an unspecified genome; no seqlengths

```r
> c13pro = c13pr[ order(ranges(c13pr)), ]
```

That’s the complete set of genes on the Watson strand of chromosome XIII. In the lceBamViews package, we do not have access to all these, but only those lying in a 100kb interval.

```r
> lim = GRanges(seqnames="Scchr13", IRanges(800000,900000), strand="+")
> c13prol = c13pro[ which(overlapsAny(c13pro , lim)), ]
```
Now that we have a set of annotation-based genomic regions, we can tabulate read counts lying in those regions and obtain an annotated matrix.

>`bamRanges(bs1) = c13pro1`
>`annotab = tabulateReads(bs1, strandmarker="+")`

## 5 Larger scale sanity check

The following plot compares read counts published with the Lee et al. (2008) paper to those computed by the methods sketched here, for all regions noted on the plus strand of chromosome XIII. Exact correspondence is not expected because of different approaches to read filtering.
6 Statistical analyses of differential expression

6.1 Using edgeR

Statistical analysis of read counts via negative binomial distributions with moderated dispersion is developed in Robinson and Smyth [2008]. The edgeR differential expression statistics are computed using regional read counts, and total library size plays a role. We compute total read counts directly (the operation can be somewhat slow for very large BAM files):

```R
> totcnts = totalReadCounts(bs1)
```

In the following demonstration, we will regard multiple lanes from the same genotype as replicates. This is probably inappropriate for this method; the original authors tested for lane effects and ultimately combined counts across lanes within strain.

```R
> library(edgeR)
> #
> # construct an edgeR container for read counts, including
> # genotype and region (gene) metadata
> #
> dgel1 = DGEList( counts=t(annotab)[-c(1,2)],
+     group=factor(bamSamples(bs1)$geno),
+     lib.size=totcnts, genes=colnames(annotab))
> #
> # compute a dispersion factor for the negative binomial model
> #
> cd = estimateCommonDisp(dgel1)
> #
> # test for differential expression between two groups
> # for each region
> #
> et12 = exactTest(cd)
> #
> # display statistics for the comparison
> #
> tt12 = topTags(et12)
> tt12
```

<table>
<thead>
<tr>
<th>Comparison of groups: rlp-isowt</th>
<th>genes</th>
<th>logFC</th>
<th>logCPM</th>
<th>PValue</th>
<th>FDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMR269W</td>
<td>YMR269W</td>
<td>4.428960</td>
<td>10.868861</td>
<td>8.540197e-10</td>
<td>2.305853e-08</td>
</tr>
<tr>
<td>YMRWdelta20</td>
<td>YMRWdelta20</td>
<td>3.218156</td>
<td>9.068794</td>
<td>2.858498e-05</td>
<td>3.858973e-04</td>
</tr>
<tr>
<td>YMR294W</td>
<td>YMR294W</td>
<td>2.293633</td>
<td>10.943004</td>
<td>3.464714e-04</td>
<td>3.118243e-03</td>
</tr>
</tbody>
</table>
An analog of the “MA-plot” familiar from microarray studies is available for this analysis. The ‘concentration’ is the log proportion of reads present in each gene, and the “log fold change” is the model-based estimate of relative abundance. In the following display we label the top 10 genes (those with smallest FDR).

```r
> plotSmear(cd, cex=.8, ylim=c(-5,5))
> text(tt12$table$logCPM, tt12$table$logFC+.15, as.character( + tt12$table$genes), cex=.65)
```

<table>
<thead>
<tr>
<th>Gene</th>
<th>LogCPM</th>
<th>LogFC</th>
<th>FDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMR286W</td>
<td>2.136146</td>
<td>10.945024</td>
<td>9.878631e-04</td>
</tr>
<tr>
<td>YMR290W-A</td>
<td>2.324576</td>
<td>9.523078</td>
<td>6.668076e-03</td>
</tr>
<tr>
<td>YMR292W</td>
<td>1.895158</td>
<td>11.384005</td>
<td>1.264757e-03</td>
</tr>
<tr>
<td>YMR312W</td>
<td>1.472817</td>
<td>11.254843</td>
<td>2.223880e-03</td>
</tr>
<tr>
<td>YMR284W</td>
<td>1.362186</td>
<td>11.272932</td>
<td>8.142389e-02</td>
</tr>
<tr>
<td>YMR267W</td>
<td>1.403464</td>
<td>11.272932</td>
<td>8.142389e-02</td>
</tr>
<tr>
<td>YMR306W</td>
<td>1.300374</td>
<td>11.565601</td>
<td>3.114871e-02</td>
</tr>
</tbody>
</table>
7 Summary

- The BAM format provides reasonably compact and comprehensive information about alignments of short reads obtained in a sequencing experiment. samtools utilities permit efficient random access to read collections of interest.

- Rsamtools brings samtools functionality into R, principally through the scanBam method, which is richly parameterized so that many details of access to and filtering of reads from BAM files can be controlled in R.

- Rsamtools defines the bamViews container for management of collections of BAM files. Read data are managed external to R; data on aligned reads can be imported efficiently, and “streaming read” models for scanning large collections of reads can be used. Many embarrassingly parallel operations can be accomplished concurrently using multicore or similar packages.

- The leeBamViews package provides small excerpts from BAM files generated after bowtie alignment of FASTQ records available through the NCBI short read archives. These excerpts can be analyzed using code shown in this vignette.

- After the count data have been generated, various approaches to inference on differential expression are available. We consider the moderated negative binomial models of edgeR above; more general variance modeling is available in the developmental DESeq package.

References


8 Session data

> sessionInfo()

R version 4.4.0 beta (2024-04-15 r86425)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 22.04.4 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

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

time zone: America/New_York
tzcode source: system (glibc)

attached base packages:
[1] stats4 stats graphics grDevices utils datasets methods
[8] base

other attached packages:
[1] edgeR_4.2.0
[3] org.Sc.sgd.db_3.19.1 AnnotationDbi_1.66.0
[5] GenomicAlignments_1.40.0 SummarizedExperiment_1.34.0
[7] MatrixGenerics_1.16.0 matrixStats_1.3.0
[9] leeBamViews_1.40.0 BSgenome_1.72.0
[11] rtracklayer_1.64.0 BiocIO_1.14.0
[13] Rsamtools_2.20.0 Biostrings_2.72.0
[15] XVector_0.44.0 GenomicRanges_1.56.0
[17] GenomeInfoDb_1.40.0 IRanges_2.38.0
[19] S4Vectors_0.42.0 Biobase_2.64.0
[21] BiocGenerics_0.50.0

loaded via a namespace (and not attached):
[1] SparseArray_1.4.0 bitops_1.0-7 RSQLite_2.3.6
[4] lattice_0.22-6 grid_4.4.0 fastmap_1.1.1