

Working with aligned nucleotides (WORK-IN-PROGRESS!)

Hervé Pagès

Last modified: January 2014; Compiled: November 12, 2019

Contents

1	Introduction	1
2	Load the aligned reads and their sequences from a BAM file. .	2
3	Compute the <i>original query sequences</i>	4
4	Mismatches, indels, and gaps.	4
5	Put the read sequences and reference sequences “side by side”	5
6	OLD STUFF (needs to be recycled/updated)	5
6.1	Load paired-end reads from a BAM file	5
7	<code>sessionInfo()</code>	8

1 Introduction

This vignette belongs to the *GenomicAlignments* package. It illustrates how to use the package for working with the nucleotide content of aligned reads.

After the reads generated by a high-throughput sequencing experiment have been aligned to a reference genome, the questions that are being asked about these alignments typically fall in two broad categories: **positional only** and **nucleotide-related**.

Positional only questions are about the position of the alignments with respect to the reference genome. Note that the position of an alignment is actually better described in terms of genomic ranges (1 range for an alignment with no gaps, 2 or more ranges for an alignment with gaps). Knowing the ranges of the alignments is sufficient to perform common tasks like *read counting* or for *computing the coverage*. *Read counting* is the process of counting the number of aligned reads per gene or exon and is typically performed in the context of a differential analysis. This task can be accomplished with the `summarizeOverlaps` function provided in the *GenomicAlignments* package and is explained in details in the “Counting reads with summarizeOverlaps” vignette (also located in this package). *Computing the coverage*

Working with aligned nucleotides (WORK-IN-PROGRESS!)

is often the preliminary step to peak detection (ChIP-seq analysis) or to a copy number analysis. It can be accomplished with the `coverage` function. See `?`coverage-methods`` for more information.

Nucleotide-related questions are about the nucleotide content of the alignments. In particular how this content compares to the corresponding nucleotides in the reference genome. These questions typically arise in the context of small genetic variation detection between one or more samples and a reference genome.

The *GenomicAlignments* package provides a suite of low- to mid-level tools for dealing with **nucleotide-related** questions about the alignments. In this vignette we illustrate their use on the single-end and paired-end reads of an RNA-seq experiment.

Note that these tools do NOT constitute a complete variant toolbox. If this is what you're looking for, other *Bioconductor* packages might be more appropriate. See the GeneticVariability and SNP views at this URL http://bioconductor.org/packages/release/BiocViews.html#___AssayDomains for a complete list of packages that deal with small genetic variations. Most of them provide tools of higher level than the tools described in this vignette. See for example the *VariantTools* and *VariantAnnotation* packages for a complete variant toolbox (including variant calling capabilities).

2 Load the aligned reads and their sequences from a BAM file

In this section, we illustrate how aligned reads and their sequences can be loaded from a BAM file. The reads we're going to use for this are paired-end reads from a published study by Zarnack et al., 2012. A subset of these reads are stored in the BAM files located in the *RNAseqData.HNRNPC.bam.chr14* data package. The package contains 8 BAM files, 1 per sequencing run:

```
> library(RNAseqData.HNRNPC.bam.chr14)
> bamfiles <- RNAseqData.HNRNPC.bam.chr14_BAMFILES
> names(bamfiles) # the names of the runs

[1] "ERR127306" "ERR127307" "ERR127308" "ERR127309" "ERR127302" "ERR127303"
[7] "ERR127304" "ERR127305"
```

Each BAM file was obtained by (1) aligning the reads (paired-end) to the full hg19 genome with TopHat2, and then (2) subsetting to keep only alignments on chr14. See `?RNAseqData.HNRNPC.bam.chr14` for more information about this data set.

As a preliminary step, we check whether the BAM files contain single- or paired-end alignments. This can be done with the `quickBamFlagSummary` utility from the *Rsamtools* package:

```
> library(Rsamtools)
> quickBamFlagSummary(bamfiles[1], main.groups.only=TRUE)
```

	group	nb of	nb of	mean / max
	of	records	unique	records per
	records	in group	QNAMEs	unique QNAME
All records.....	A	800484	393300	2.04 / 10
o template has single segment....	S	0	0	NA / NA

Working with aligned nucleotides (WORK-IN-PROGRESS!)

```
o template has multiple segments. M | 800484 | 393300 | 2.04 / 10
- first segment..... F | 400242 | 393300 | 1.02 / 5
- last segment..... L | 400242 | 393300 | 1.02 / 5
- other segment..... O | 0 | 0 | NA / NA
```

Note that (S, M) is a partitioning of A, and (F, L, O) is a partitioning of M. Indentation reflects this.

This confirms that all the alignments in the 1st BAM file (run ERR127306) are paired-end. This means that we should preferably load them with the `readGAlignmentPairs` function from the *GenomicAlignments* package. However for the purpose of keeping things simple, we will ignore the pairing for now and load only the alignments corresponding to the first segment of the pairs. We will use the `readGAlignments` function from the *GenomicAlignments* package for this, together with a *ScanBamParam* object for the filtering. See `?ScanBamParam` in the *Rsamtools* package for the details.

Furthermore, while preparing the *ScanBamParam* object to perform the filtering, we'll also get rid of the PCR or optical duplicates (flag bit 0x400 in the SAM format, see the SAM Spec ¹ for the details), as well as reads not passing quality controls (flag bit 0x200 in the SAM format).

¹<http://samtools.sourceforge.net/>

Finally we also request the read sequences (a.k.a. the *segment sequences* in the SAM Spec, stored in the SEQ field) via the *ScanBamParam* object:

```
> flag1 <- scanBamFlag(isFirstMateRead=TRUE, isSecondMateRead=FALSE,
+                      isDuplicate=FALSE, isNotPassingQualityControls=FALSE)
> param1 <- ScanBamParam(flag=flag1, what="seq")
```

We're now ready to load the alignments and their sequences. Note that we use `use.names=TRUE` in order to also load the *query names* (a.k.a. the *query template names* in the SAM Spec, stored in the QNAME field) from the BAM file. `readGAlignments` will use them to set the names of the returned object:

```
> library(GenomicAlignments)
> gal1 <- readGAlignments(bamfiles[1], use.names=TRUE, param=param1)
```

This returns a *GAlignments* object. The read sequences are stored in the `seq` metadata column of the object:

```
> mcols(gal1)$seq
A DNAStringSet instance of length 400242
width seq
[1] 72 TGAGAATGATGATTTCCAATTTTCATCCATGT...GACATGAACATCATTTTTTATGGCTGCAT
[2] 72 CCCAGGTATACACTGGACTCCAGGTGGACA...GTTGGATACACACTCAAGGTGGACACCAG
[3] 72 CATAGATGCAAGAATCCTCAATCAAATACTA...TTCAACAGCACATTAAAAAGATAACTACCA
[4] 72 TGCTGGTGCAGGATTATTCTACTAAGCAAT...ATCAAATCCACTTTCTTATCTCAGGAATCAG
[5] 72 CAGGAGGTAGGCTGTGCGTTCAGCAGTTGGT...GGTACTGGTTGATCACCTTGACTGTCTGGTC
... ...
[400238] 72 GGGAGGCCCTTTATATAACCATCAGGTCTTG...CTCACTAATAGGATAAAAGCATGGAGAGAAC
[400239] 72 CCTGAGAGCCCTTGTCTGCTGAGCACCTC...GAGCGCCCTCTGGTGTCTGATCACTCTCTG
[400240] 72 CAACTTTTATTCTTAAACACAAGACATTCC...CTGTTCTCAGGTGAGCTGTCGAGCAGGGAGG
[400241] 72 CAAAGCTGGATGTGCTAGTGTTTTTATCAG...CCGTAATAAGAGCATGTGTGTTTTGCTGCC
```

```
[400242]    72  CATGACTTGATGGCTGGAACAAATACATTTA...CTCCAATACTAGCCTTTGCCATACAGTATTT
```

3 Compute the *original query sequences*

Because the BAM format imposes that the read sequence is “reverse complemented” when a read aligns to the minus strand, we need to “reverse complement” it again to restore the *original query sequences* (i.e. the sequences before alignment, that is, as they can be seen in the FASTQ file assuming that the aligner didn't perform any hard-clipping on them):

```
> oqseq1 <- mcols(gal1)$seq
> is_on_minus <- as.logical(strand(gal1) == "-")
> oqseq1[is_on_minus] <- reverseComplement(oqseq1[is_on_minus])
```

Because the aligner used to align the reads can report more than 1 alignment per read (i.e. per sequence stored in the FASTQ file), we shouldn't expect the names of `gal1` to be unique:

```
> is_dup <- duplicated(names(gal1))
> table(is_dup)
```

```
is_dup
FALSE   TRUE
393300  6942
```

However, sequences with the same *query name* should correspond to the same *original query* and therefore should be the same. Let's do a quick sanity check:

```
> dup2unq <- match(names(gal1), names(gal1))
> stopifnot(all(oqseq1 == oqseq1[dup2unq]))
```

Finally, let's reduce `oqseq1` to one *original query sequence* per unique *query name* (like in the FASTQ file containing the 1st end of the unaligned reads):

```
> oqseq1 <- oqseq1[!is_dup]
```

4 Mismatches, indels, and gaps

Because the aligner possibly tolerated a small number of mismatches, indels, and/or gaps during the alignment process, the sequences in `mcols(gal1)$seq` generally don't match exactly the reference genome.

The information of where indels and/or gaps occur in the alignments is represented in the CIGAR strings. Let's have a look at these string. First the most frequent cigars:

```
> head(sort(table(cigar(gal1)), decreasing=TRUE))
      72M 35M123N37M 64M316N8M 38M670N34M 18M123N54M 2M131N70M
301920      134      134      133      119      96
```

Then a summary of the total number of insertions (I), deletions (D), and gaps (N):

Working with aligned nucleotides (WORK-IN-PROGRESS!)

```
> colSums(cigarOpTable(cigar(gal1)))
```

	M	I	D	N	S	H	P	=
28815928		1496	818	330945002	0	0	0	0
X								
0								

This tells us that the aligner that was used supports indels (I/D) and junction reads (N).

Finally we count and summarize the number of gaps per alignment:

```
> table(njunc(gal1))
```

0	1	2	3
303622	94557	2052	11

Some reads contain up to 3 gaps (i.e. span 3 splice junctions).

5 Put the read sequences and reference sequences “side by side”

TODO (with `sequenceLayer`)

6 OLD STUFF (needs to be recycled/updated)

6.1 Load paired-end reads from a BAM file

BAM file `untreated3_chr4.bam` (located in the `pasillaBamSubset` data package) contains paired-end reads from the “Pasilla” experiment and aligned against the dm3 genome (see `?untreated3_chr4` in the `pasillaBamSubset` package for more information about those reads). We use the `readGAlignmentPairs` function to load them into a `GAlignmentPairs` object:

```
> library(pasillaBamSubset)
> flag0 <- scanBamFlag(isDuplicate=FALSE, isNotPassingQualityControls=FALSE)
> param0 <- ScanBamParam(flag=flag0)
> U3.galp <- readGAlignmentPairs(untreated3_chr4(), use.names=TRUE, param=param0)
> head(U3.galp)
```

`GAlignmentPairs` object with 6 pairs, strandMode=1, and 0 metadata columns:

	seqnames	strand	:	ranges	--	ranges
	<Rle>	<Rle>	:	<IRanges>	--	<IRanges>
SRR031715.1138209	chr4	+	:	169-205	--	326-362
SRR031714.756385	chr4	+	:	943-979	--	1086-1122
SRR031714.2355189	chr4	+	:	944-980	--	1119-1155
SRR031714.5054563	chr4	+	:	946-982	--	986-1022
SRR031715.1722593	chr4	+	:	966-1002	--	1108-1144
SRR031715.2202469	chr4	+	:	966-1002	--	1114-1150

Working with aligned nucleotides (WORK-IN-PROGRESS!)

```
-----  
seqinfo: 8 sequences from an unspecified genome
```

The `show` method for *GAlignmentPairs* objects displays two ranges columns, one for the *first* alignment in the pair (the left column), and one for the *last* alignment in the pair (the right column). The strand column corresponds to the strand of the *first* alignment.

```
> head(first(U3.galp))  
  
GAlignments object with 6 alignments and 0 metadata columns:  
      seqnames strand      cigar  qwidth  start    end  
      <Rle>   <Rle> <character> <integer> <integer> <integer>  
SRR031715.1138209 chr4      +      37M      37     169     205  
SRR031714.756385  chr4      +      37M      37     943     979  
SRR031714.2355189 chr4      +      37M      37     944     980  
SRR031714.5054563 chr4      +      37M      37     946     982  
SRR031715.1722593 chr4      +      37M      37     966    1002  
SRR031715.2202469 chr4      +      37M      37     966    1002  
      width  njunc  
      <integer> <integer>  
SRR031715.1138209      37      0  
SRR031714.756385      37      0  
SRR031714.2355189      37      0  
SRR031714.5054563      37      0  
SRR031715.1722593      37      0  
SRR031715.2202469      37      0  
-----  
seqinfo: 8 sequences from an unspecified genome  
  
> head(last(U3.galp))  
  
GAlignments object with 6 alignments and 0 metadata columns:  
      seqnames strand      cigar  qwidth  start    end  
      <Rle>   <Rle> <character> <integer> <integer> <integer>  
SRR031715.1138209 chr4      -      37M      37     326     362  
SRR031714.756385  chr4      -      37M      37    1086    1122  
SRR031714.2355189 chr4      -      37M      37    1119    1155  
SRR031714.5054563 chr4      -      37M      37     986    1022  
SRR031715.1722593 chr4      -      37M      37    1108    1144  
SRR031715.2202469 chr4      -      37M      37    1114    1150  
      width  njunc  
      <integer> <integer>  
SRR031715.1138209      37      0  
SRR031714.756385      37      0  
SRR031714.2355189      37      0  
SRR031714.5054563      37      0  
SRR031715.1722593      37      0  
SRR031715.2202469      37      0  
-----  
seqinfo: 8 sequences from an unspecified genome
```

Working with aligned nucleotides (WORK-IN-PROGRESS!)

According to the SAM format specifications, the aligner is expected to mark each alignment pair as *proper* or not (flag bit 0x2 in the SAM format). The SAM Spec only says that a pair is *proper* if the *first* and *last* alignments in the pair are “properly aligned according to the aligner”. So the exact criteria used for setting this flag is left to the aligner.

We use `isProperPair` to extract this flag from the `GAlignmentPairs` object:

```
> table(isProperPair(U3.galp))  
FALSE TRUE  
29581 45828
```

Even though we could do *overlap encodings* with the full object, we keep only the *proper* pairs for our downstream analysis:

```
> U3.GALP <- U3.galp[isProperPair(U3.galp)]
```

Because the aligner used to align those reads can report more than 1 alignment per *original query template* (i.e. per pair of sequences stored in the input files, typically 1 FASTQ file for the *first* ends and 1 FASTQ file for the *last* ends), we shouldn't expect the names of `U3.GALP` to be unique:

```
> U3.GALP_names_is_dup <- duplicated(names(U3.GALP))  
> table(U3.GALP_names_is_dup)  
U3.GALP_names_is_dup  
FALSE TRUE  
43659 2169
```

Storing the *query template names* in a factor will be useful:

```
> U3.uqnames <- unique(names(U3.GALP))  
> U3.GALP_qnames <- factor(names(U3.GALP), levels=U3.uqnames)
```

as well as having the mapping between each *query template name* and its first occurrence in `U3.GALP_qnames`:

```
> U3.GALP_dup2unq <- match(U3.GALP_qnames, U3.GALP_qnames)
```

Our reads can have up to 1 gap per end:

```
> head(unique(cigar(first(U3.GALP))))  
[1] "37M" "6M58N31M" "25M56N12M" "19M62N18M" "29M222N8M" "9M222N28M"  
> head(unique(cigar(last(U3.GALP))))  
[1] "37M" "19M58N18M" "12M58N25M" "27M2339N10M" "29M2339N8M"  
[6] "9M222N28M"  
> table(njunc(first(U3.GALP)), njunc(last(U3.GALP)))  
      0      1  
0 44510 596  
1 637 85
```

Like for our single-end reads, the following tables indicate that indels were not allowed/supported during the alignment process:

Working with aligned nucleotides (WORK-IN-PROGRESS!)

```
> colSums(cigarOpTable(cigar(first(U3.GALP))))  
      M      I      D      N      S      H      P      =      X  
1695636      0      0 673919      0      0      0      0      0  
  
> colSums(cigarOpTable(cigar(last(U3.GALP))))  
      M      I      D      N      S      H      P      =      X  
1695636      0      0 630395      0      0      0      0      0
```

7 sessionInfo()

```
> sessionInfo()  
  
R version 3.6.1 (2019-07-05)  
Platform: x86_64-w64-mingw32/x64 (64-bit)  
Running under: Windows Server 2012 R2 x64 (build 9600)  
  
Matrix products: default  
  
locale:  
[1] LC_COLLATE=C  
[2] LC_CTYPE=English_United States.1252  
[3] LC_MONETARY=English_United States.1252  
[4] LC_NUMERIC=C  
[5] LC_TIME=English_United States.1252  
  
attached base packages:  
[1] stats4      parallel  stats      graphics  grDevices  utils      datasets  
[8] methods     base  
  
other attached packages:  
[1] RNAseqData.HNRNPC.bam.chr14_0.24.0  
[2] BSgenome.Dmelanogaster.UCSC.dm3_1.4.0  
[3] BSgenome_1.54.0  
[4] rtracklayer_1.46.0  
[5] TxDb.Dmelanogaster.UCSC.dm3.ensGene_3.2.2  
[6] GenomicFeatures_1.38.0  
[7] AnnotationDbi_1.48.0  
[8] pasillaBamSubset_0.24.0  
[9] GenomicAlignments_1.22.1  
[10] Rsamtools_2.2.1  
[11] Biostrings_2.54.0  
[12] XVector_0.26.0  
[13] SummarizedExperiment_1.16.0  
[14] DelayedArray_0.12.0  
[15] BiocParallel_1.20.0  
[16] matrixStats_0.55.0  
[17] Biobase_2.46.0  
[18] GenomicRanges_1.38.0
```


Working with aligned nucleotides (WORK-IN-PROGRESS!)

```
[19] GenomeInfoDb_1.22.0
[20] IRanges_2.20.0
[21] S4Vectors_0.24.0
[22] BiocGenerics_0.32.0
```

loaded via a namespace (and not attached):

```
[1] Rcpp_1.0.3          lattice_0.20-38      prettyunits_1.0.2
[4] assertthat_0.2.1    zeallot_0.1.0        digest_0.6.22
[7] BiocFileCache_1.10.2 R6_2.4.0             backports_1.1.5
[10] RSQLite_2.1.2       evaluate_0.14        http_1.4.1
[13] pillar_1.4.2        zlibbioc_1.32.0      rlang_0.4.1
[16] progress_1.2.2      curl_4.2             blob_1.2.0
[19] Matrix_1.2-17       rmarkdown_1.16       stringr_1.4.0
[22] RCurl_1.95-4.12     bit_1.1-14           biomaRt_2.42.0
[25] compiler_3.6.1      xfun_0.11            pkgconfig_2.0.3
[28] askpass_1.1         htmltools_0.4.0      openssl_1.4.1
[31] tidyselect_0.2.5    tibble_2.1.3         GenomeInfoDbData_1.2.2
[34] XML_3.98-1.20       crayon_1.3.4         dplyr_0.8.3
[37] dbplyr_1.4.2        bitops_1.0-6         rappdirs_0.3.1
[40] grid_3.6.1          DBI_1.0.0            magrittr_1.5
[43] stringi_1.4.3       vctrs_0.2.0          BiocStyle_2.14.0
[46] tools_3.6.1         bit64_0.9-7          glue_1.3.1
[49] purrr_0.3.3         hms_0.5.2            yaml_2.2.0
[52] BiocManager_1.30.9 memoise_1.1.0        knitr_1.26
```