Introduction to Sequences : Reading and Manipulating Short Reads

Valerie Obenchain and Martin Morgan

Fred Hutchinson Cancer Research Center

9-10 December, 2010

Contents

1 Introduction **1**

2 Aligned read input **2**
2.1 Preliminaries 2
2.2 `readAligned` and the *AlignedRead* class 3
2.3 Base calls 4
2.4 Alignment 7
2.5 Subsets and filters 8
2.6 Cautions 9

3 Quality assessment **10**
3.1 Generating a QA report 10

4 Session information **11**

1 Introduction

This practical uses the *ShortRead* package to input aligned and other short read data files, and illustrates some of the available sequence manipulation and quality assessment tools. Activities during the lab are posed as exercises. So, as a first exercise:

Exercise 1
Start an R session, and load the ShortRead package.

```
> library(ShortRead)
> packageDescription("ShortRead")$Version
```

```
[1] "1.8.2"
```

Confirm that the version of your package is at least as recent as the version in this document. Seek assistance from one of the course assistants if you need help getting the current version of ShortRead.

The course also requires access to sample data. We will be using yeast data from an RNA-seq experiment. See `browseVignette("SeattleIntro2010")` for experiment details.

Exercise 2
Load the SeattleIntro2010 package. Use

```
> library(SeattleIntro2010)
```

2 Aligned read input

This section illustrates input of aligned reads. It focuses on aligned reads in *BAM* format; reading data produced by software such as *MAQ* or *Bowtie* is described in the *ShortRead* 'Overview' vignette and on the `readAligned` help page.

2.1 Preliminaries

Vendor and third-party software is likely to process raw images, base calling (*Rolexa* is a *Bioconductor* package providing alternative base calling), and alignment to a reference genome (see *BSgenome* and the `matchPDict` function of *Biostrings*). The following work flow starts with reads aligned with *BWA* software in *BAM* format.

We'll start by creating a variable, `extdataDir`, containing the directory holding the sample data, and check to make sure that we have defined the right path.

```
> extdataDir <-
+     system.file("extdata", package="SeattleIntro2010")
> file.exists(extdataDir)
```

```
[1] TRUE
```

```
> stopifnot(file.exists(extdataDir))
```

A key functionality provided by *ShortRead* is input of a diversity of file types, both of files from the Illumina pipeline and from other software and in formats appropriate for other technologies. The interface to these input functions is meant to facilitate reading one or more files into a single object. The interface is like that for `list.files`: provide a directory path where relevant files are to be found, plus a regular expression to select the files you are interested in.

Exercise 3
Use `list.files` to display all files in the `extdataDir` directory.

```
> list.files(extdataDir)
```

```
 [1] "GeneRange.bed"
 [2] "SRR002051.chrI-V.bam"
 [3] "SRR002051.chrI-V.bam.bai"
 [4] "SRR002051.reads1-50k.fastq"
 [5] "SRR002051.reads1-50k.fastq.NOTE"
```

```
 [6] "exprsMat.csv"
 [7] "mm9_knownGene.sqlite"
 [8] "pData.csv"
 [9] "sacCer2_sgdGene.sqlite"
[10] "subData.bed"
```

We'll select just one file to use in this analysis, based on files matching the regular expression `SRR002051.chrI-V.bam$`. The use of the special character '$' ensures that we don't also match the *BAM* index file (extention `.bai`) in the same directory. Other inputs (e.g., Solexa, *MAQ*, *Bowtie*, or *BWA* files) are also supported.

Exercise 4
We'll use an abbreviated file for most parts of this lab. The abbreviated file contains chromosomes 1 through 5 from a single lane of an Illumina run. The name of the file is `SRR002051.chrI-V.bam`. We will match this 'pattern' and check that we have specified the pattern appropriately.

```
> pattern <- "SRR002051.chrI-V.bam$"
> list.files(extdataDir, pattern)
```

```
[1] "SRR002051.chrI-V.bam"
```

2.2 `readAligned` and the *AlignedRead* class

The `readAligned` function can be used to input aligned reads. The first argument is a directory path where alignment files are to be found. The second argument is the regular expression to select files to be read. An optional third argument allows the user to specify which type of file is to be read in.

Exercise 5
Use `readAligned` to read in the data, specifying `BAM` as the type of file.

```
> aln <- readAligned(extdataDir, pattern, type="BAM")
```

See the help page for `readAligned` for additional details about supported file types.

What does `readAligned` input? It inputs the short read sequences and base call qualities, and the chromosome, position, and strand information associated with short read alignments. This information is expected to be provided by all short read alignemnt software.

Exercise 6
Display the object we have read in.

```
> aln
```

```
class: AlignedRead
length: 446075 reads; width: 33 cycles
chromosome: chrI chrI ... chrV chrV
position: 11 1062 ... 576545 576836
strand: - + ... - -
alignQuality: NumericQuality
alignData varLabels: flag
```

There are 446075 reads in the object, each read consisting of 33 nucleotides. View the first several reads and query information about, e.g., the number of reads that align to each strand, or to each chromosome.

```
> head(sread(aln))

  A DNAStringSet instance of length 6
    width seq
[1]    33 TGTGGTGTGTGGTGTGGTGTGTGGGTGTGTGGG
[2]    33 TGCATCTTTAATCTTGTATGTTACACTACTCAT
[3]    33 TTAAATAACGTACCTATCACAGTATCGTCTTGA
[4]    33 TCTAAATGAGAGTTTGGTACCATGACTTTTAAC
[5]    33 ACAAGCAACTCATAATTTAAGTGGATATCTTTT
[6]    33 AGTTTTCTCAAACGCTTGATAGCATGATTTGAT

> table(strand(aln), useNA="ifany")

     +      -      *
215256 230819      0

> table(chromosome(aln), useNA="ifany")

2micron    chrI   chrII  chrIII   chrIV   chrIX    chrM
      0   34753  103268   30551  213176       0       0
   chrV   chrVI  chrVII chrVIII    chrX   chrXI  chrXII
  64327       0       0       0       0       0       0
chrXIII  chrXIV   chrXV  chrXVI
      0       0       0       0
```

Notice how elements of the `aln` *object are extracted using* accessors *such as* `sread` *and* `strand`*; these are described on the help page for the class of the* `aln` *object (indicated in the display of* `aln`*, above, as class AlignedRead); note that the help page refers to the help page for* `accessors` *to enumerate additional ways of accessing the data.*

The `strand` function returns a factor with three levels. The first two describe reads aligned to the plus and minus strands, the third (`*`) is available for successful alignments where strand information is irrelevant. This *BAM* file was created by filtering out reads that did not align to chromosomes I-V so there are no `NA` values in the data. The approximately equal distribution of reads between the positive and negative strands indicate there is no bias.

Aligned reads contain several different kinds of information about 'quality'. Individual bases are assessed for quality during base calling. These 'raw' base qualities are 'calibrated' during alignment. The alignments themselves also have qualities associated with them, with the details of alignment quality differing between alignment algorithms.

2.3 Base calls

Exercise 7

Use `alphabetFrequency` *to summarize nucleotide use in the short reads in* `aln`*.*

```
> alphabetFrequency(sread(aln), collapse=TRUE, baseOnly=TRUE,
+                   as.prob=TRUE)

           A            C            G            T
0.3640116233 0.1645332097 0.1602367451 0.3103772806
       other
0.0008411413
```

The frequency of 'other' (i.e., uncalled) nucleotides is typically $< 1\%$.

The expectation is that sequences start at essentially arbitrary locations in the sequenced DNA so nucloetide frequencies should be approximately constant over the cycles (i.e., an A is expected as frequently at the beginning of the sequence as at the end).

Exercise 8
Use `alphabetByCycle` *to extract the number of each nucleotide A, C, T, G, sequenced at each cycle. Plot the transpose of the matrix using* `matplot`*, to see how nucleotide counts change across cycles.*

```
> abc <- alphabetByCycle(sread(aln))
> dim(abc)

[1] 17 33

> abc[1:4,1:5]

        cycle
alphabet   [,1]   [,2]   [,3]   [,4]   [,5]
       A 187244 175248 174714 179343 161588
       C  75581  68185  59399  61533  62913
       G  48303  59949  79958  68827  60694
       T 134947 142684 132004 136370 160879

> matplot(t(abc[1:4,]), type="l", lty=c(1:4),
+ col=c("green","red","black","blue"), xlab="Cycle", ylab="Count")
> legend("top", dimnames(abc)$alphabet[1:4], lty=c(1:4),
+ col=c("green","red","black","blue"), bty="n", horiz=T)
```

There are a number of possible contributors to cycle-dependent nucleotide frequencies. Early experiments such as this frequently suffered from inadequate reagent volume and nucleotide-specific differential accumulation of fluorescent dyes.

Exercise 9
Retrieve calibrated base quality from `aln`*.*

```
> head(quality(aln))

class: FastqQuality
quality:
  A BStringSet instance of length 6
    width seq
```

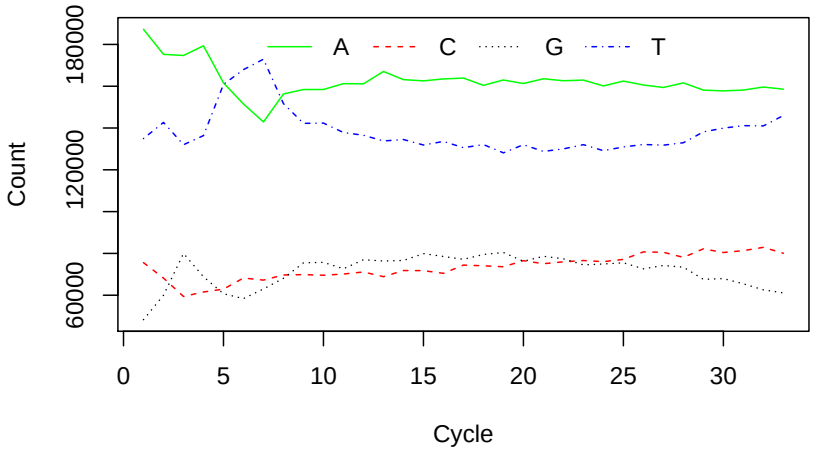

Figure 1: Nucleotide frequency per cycle

```
[1]     33 IIIIIIIIIIIIIIIIIIE/8A9I+2I20,4(
[2]     33 IIIIIIIII1IIIIIEFII1%)<1<:H&+776
[3]     33 IIIIIIIIIIIIIIIIIIIIIEIIIIIII&-
[4]     33 IIIIIIIII8IIIIIIIIG:=5*>I9'1I):
[5]     33 IIIIIIIII9IIIIIIIID4*I?203(;I3"&
[6]     33 IIIIIIIIIIIIIIIIIIII-II;III%:I
```

These qualities are string-encoded $-10\log_{10}$ *probabilities. The encoding in this case follows a convention established by Illumina. The details of the encoding can be obtained by querying* `quality(aln)` *for its* `alphabet`*; the letter* `A` *corresponds to a* $-10log_{10}$ *score of 1.*

Numeric values are readily retrieved as a matrix, with rows corresponding to reads and columns to cycles. Computations can then be performed on them, e.g., to determine average calibrated quality scores as a function of cycle.

```
> alf <- alphabet(quality(aln))
> m <- as(quality(aln), "matrix")
> colMeans(m)

 [1] 39.41089 39.35913 39.24368 39.18939 39.08526 38.99672
 [7] 38.79983 38.30618 38.02001 37.85870 36.86351 37.08458
[13] 36.79942 36.15783 35.78240 35.37136 34.56440 34.05317
[19] 32.41027 32.33367 31.36152 30.40437 29.60604 27.79380
[25] 27.03094 26.24773 25.15012 23.75580 22.54234 21.28405
[31] 20.22223 19.00269 18.39202
```

Exercise 10
It may be of interest to identify the GC-rich reads in our sample.

```
> res <- alphabetFrequency(sread(aln), baseOnly=TRUE)
> head(res)
```

```
      A C  G  T other
[1,]  0 0 20 13     0
[2,]  8 7  3 15     0
[3,] 11 7  4 11     0
[4,] 10 5  6 12     0
[5,] 12 5  4 12     0
[6,]  9 5  6 13     0
```

```
> gcContent <- rowSums(res[,c("G", "C")]) / rowSums(res)
> print(histogram(gcContent, xlab="Percent GC",
+                 ylab="Percent of total reads"))
> gcIdx <- gcContent > 0.6
> aln[gcIdx]
```

```
class: AlignedRead
length: 2117 reads; width: 33 cycles
chromosome: chrI chrI ... chrV chrV
position: 11 11802 ... 576096 576240
strand: - - ... + -
alignQuality: NumericQuality
alignData varLabels: flag
```

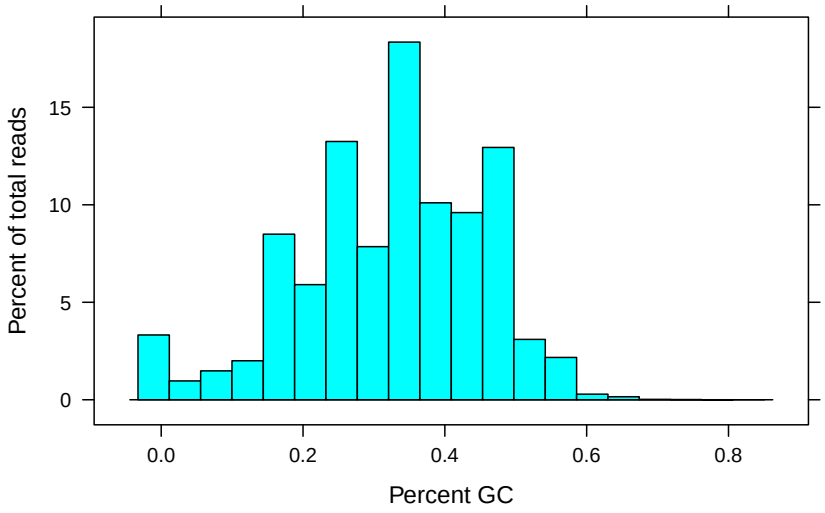

Figure 2: GC-rich reads

2.4 Alignment

Alignment qualities are accessible with `alignQuality`. This returns an object that can contain quality scores in different formats; to extract the actual quality

scores, use `quality`. Reads failing to align or to align in multiple locations have an alignment quality of 0.

Exercise 11
Retrieve the alignment quality scores, determine how many align poorly, and visualize the distribution (Figure 3) of scores.

```
> alignQuality(aln)
```

```
class: NumericQuality
quality: 37 0 ... 0 0 (446075 total)
```

```
> q <- quality(alignQuality(aln))
> sum(q==0)
```

```
[1] 64167
```

```
> print(densityplot(q[q>1], plot.points=FALSE,
+                   xlab="Alignment quality"))
```

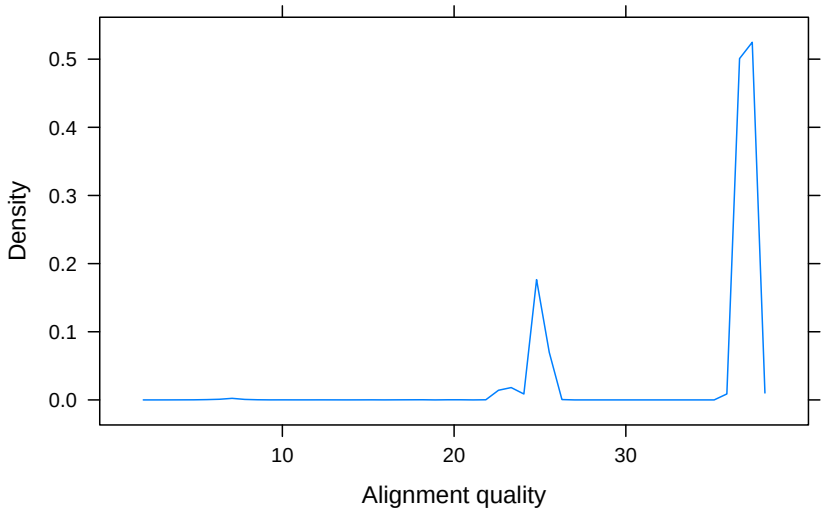

Figure 3: Alignment quality

Alignment algorithms produce information in addition to basic data about chromosome, position, and strand alignment. The exact content varies between algorithms, and is available with `alignData`. `alignData` returns an *AlignedDataFrame* object that contains these data and a metadata description of them.

2.5 Subsets and filters

A very common operation is to reduce the number of reads used for subsequent analysis. This can be done in a coordinated fashion by creating a subset of `aln`.

Exercise 12
Select just the aligned reads from chromosome III and V.

```
> chrfilt <- chromosomeFilter("chrIII|chrV")
> aln[chrfilt(aln)]
```

```
class: AlignedRead
length: 94878 reads; width: 33 cycles
chromosome: chrIII chrIII ... chrV chrV
position: 876 1224 ... 576545 576836
strand: - + ... - -
alignQuality: NumericQuality
alignData varLabels: flag
```

The filter functions built-in to *ShortRead* use a 'factory' pattern to create instances of each filter that 'remember' how the filters were created. For instance, `chromosomeFilter("chrV")` creates an instance of the chromosome filter to select only chromosomes matching `chrV`.

Exercise 13
As an advanced example, the following filter subsamples a (user-specified) number of reads. The `samplingFilter` function uses the factory pattern, so filters created with it remember how many reads to sample.

```
> samplingFilter <- function(sampleSize) {
+     srFilter(function(x) {
+         idx <- seq_len(length(x))
+         idx %in% sample(idx, sampleSize)
+     }, name="Demo sampling filter")
+ }
> sample100 <- samplingFilter(100)
> aln[sample100(aln)]
```

```
class: AlignedRead
length: 100 reads; width: 33 cycles
chromosome: chrI chrI ... chrV chrV
position: 64198 71854 ... 538467 548175
strand: - - ... - +
alignQuality: NumericQuality
alignData varLabels: flag
```

2.6 Cautions

There are several confusing areas associated with reading data aligned with various software packages. (1) Some alignment programs and genome resources start numbering nucleotides of the subject sequence at 0, whereas others start at 1. (2) Some alignment programs report matches on the minus strand in terms of the 'left-most' position of the read (i.e., the location of the 3' end of the aligned read), whereas other report 'five-prime' matches (i.e., in terms of the 5' end of the read), regardless of whether the alignment is on the plus or minus strand. (3) Some alignment programs reverse complement the sequence of reads aligned to the minus strand. (4) Base qualities are sometimes encoded

as character strings, but the encoding differs between 'fastq' and 'solexa fastq'. It seems that all combinations of these choices are common 'in the wild'.

The help page for `readAligned` attempts to be explicit about how reads are formatted. Briefly:

- Subject sequence nucleotides are numbered starting at 1, rather than zero. `readAligned` adjusts the coordinate system of input reads if necessary (e.g., when reading MAQ alignments).
- ELAND and Bowtie alignments on the minus strand are reported in 'left-most' coordinates systems.
- ELAND and Bowtie alignments on the minus strand are not reverse complemented.

Alignment programs sometimes offer the opportunity to customize output; such customization needs to be accommodated when reads are input using *ShortRead*.

3 Quality assessment

This part of the course addresses *ShortRead* facilities for assessing quality, primarily of *BAM* data. The *ShortRead* functionality is mean to complement rather than replace QA tools provided by manufacturer pipelines.

The data we are using is from an RNA-seq experiment, using either random hexamer or oligo(dT) primers to isolate RNA. The data consist of six lanes of an early Solexa run, three from random hexamer, three from oligo(dT). In each group of three, there is an original, biological replicate and technical replicate (of original).

3.1 Generating a QA report

Creating a QA report is a two-step process. The first step is to visit necessary files to collate information in a compact representation. The second step is to present the information in a useful format.

The `qa` function collates information for the QA report. It visits each _fastq.bam file, and extracts information on reads and their qualities. Evaluation of the function is straight-forward, e.g., `qa <- qa(extdataDir, pattern, type="BAM")`. The process of collating files can be time consuming (each export file must be parsed, taking 3-4 minutes per file) and memory intensive (lanes are processed independently of one another, but processing a full lane consumes 2-3 GB of memory). The return value of the `qa` function is actually quite compact, and easy to work with.

Exercise 14
Rather than collating information during the lab, we load the data from a previously stored instance. The components of the qa object displayed with names(qa) give a sense of what has been collated during the call to qa but are primarily for internal use.

```
> data("qa", package="SeattleIntro2010")
> names(qa)
```

```
 [1] "readCounts"            "baseCalls"
 [3] "readQualityScore"      "baseQuality"
 [5] "alignQuality"          "frequentSequences"
 [7] "sequenceDistribution" "perCycle"
 [9] "perTile"               "adapterContamination"
```

One feature of *ShortRead* that can speed up this stage of the operation is the use of clustered computer resources and the *Rmpi* or *mullticore* package; `qa` uses the `srapply` function to automatically detect and distribute collation tasks across pre-established nodes. This is outlined in more detail in the *ShortRead* Vignette.

Primer and replicate type information can be seen in the hitspergene data file included with the course package.

```
> data("hitspergene", package="SeattleIntro2010")
> elementMetadata(hitspergene)
```

```
DataFrame with 6 rows and 3 columns
              Sample   Replicate         SRR
         <character> <character> <character>
SRR002062          dT  Biological   SRR002062
SRR002051          dT    Original   SRR002051
SRR002064          dT   Technical   SRR002064
SRR002058          RH  Biological   SRR002058
SRR002059          RH    Original   SRR002059
SRR002061          RH   Technical   SRR002061
```

The QA information collated from all six lanes is summarized into an html report using the `report` function. The command to create the report is `rpt <- report(qa, dest=tempfile())`. This creates an HTML file at the location specified by the argument `dest`.

Exercise 15
Create a report from the `qa` *object loaded in the previous exercise. Do this with the command*

```
> rpt <- report(qa, dest=tempfile())
```

View the report in your browser with

```
> browseURL(rpt)
```

The QA report provides summary statistics about the numbers of reads and alignments, base calls and qualities, characteristics of per-lane and per-tile read quality, and other information. The QA report is self-documenting, providing a narrative description of each section. Take some time to look over the report and observe the behavior in the 'Cycle-specific base calls and read quality' section.

4 Session information

- R version 2.12.0 Patched (2010-11-28 r53696), `i386-pc-mingw32`

- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: ALL 1.4.7, AnnotationDbi 1.12.0, BSgenome 1.18.2, BSgenome.Scerevisiae.UCSC.sacCer2 1.3.16, Biobase 2.10.0, Biostrings 2.18.2, DBI 0.2-5, DESeq 1.2.1, GO.db 2.4.5, GenomicFeatures 1.2.3, GenomicRanges 1.2.2, IRanges 1.8.7, RCurl 1.5-0.1, RSQLite 0.9-4, Rsamtools 1.2.1, SeattleIntro2010 0.0.38, ShortRead 1.8.2, akima 0.5-4, biomaRt 2.6.0, bitops 1.0-4.1, genefilter 1.32.0, hgu95av2.db 2.4.5, lattice 0.19-13, limma 3.6.9, locfit 1.5-6, multtest 2.6.0, org.Hs.eg.db 2.4.6, org.Sc.sgd.db 2.4.6, rtracklayer 1.10.6
- Loaded via a namespace (and not attached): KernSmooth 2.23-4, MASS 7.3-9, RColorBrewer 1.0-2, XML 3.2-0.2, annotate 1.28.0, geneplotter 1.28.0, grid 2.12.0, hwriter 1.3, splines 2.12.0, survival 2.36-2, tools 2.12.0, xtable 1.5-6