I/0 and Quality Assessment using ShortRead

Martin Morgan
Fred Hutchinson Cancer Research Center
Seattle, WA 98008

22 January 2009

Contents

1 Introduction

This portion of the course uses the ShortRead package to input aligned and other short read data files, and illustrates some of the available Solexa-based 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.

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

```
[1] "1.1.36"
```

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.

Exercise 2
Copy the data from the distribution media to your local hard drive. In R change the working directory to point to the data location, along the lines of

```
> setwd("c:/Documents and Settings/mtmorgan/Desktop/CourseData")
```

and confirm that the files have been copied correctly.

2 Aligned read input

This section illustrates input of aligned reads. It focuses on aligned reads produced by the Solexa Genome Analyzer ELAND software; reading data produced by software such as MAQ or Bowtie (Bowtie or by other short read technologies is supported in the development version of ShortRead) is described in the ShortRead 'Overview' vignette and on the `readAligned` help page.

2.1 *SolexaPath*: navigating Solexa output

This section introduces a way to conveniently navigate the hierarchy of files produced by ELAND; this simplifies subsequent activities, but the full ELAND output is not required to use ShortRead.

Solexa software processes data in a pipeline. Raw images are extracted to image intensity files by the Firecrest software compoent. Image intensity files are summarized as base calls using the Bustard base caller. Subsequent analysis is performed by a diversity of software components called Gerald; one example of a Gerald program is ELAND, a whole-genome aligner.

The pipepline provides (or can be configured to provide – the people running the machine have quite a bit of control over this) exquisite detail about each stage of the process. For instance, Firecrest scans each lane of a flow cell as 300 *tiles*, arranged in three serpentine columns. The Firecrest output is summarized in two different file types for each tile, so there are $2 \times 300 \times 8 = 4800$ files produced by Firecrest alone.

A portion of a file hierarchy is provided as course data.

Exercise 3
Consult the help page for `SolexaPath`, and create an instance of this object, e.g.,

```
> sp <- SolexaPath("extdata/ELAND/080828_HWI-EAS88_0003")
```

This command scans the file system rooted at the specified path (the final directory name given above is a typical top level name for a Solexa run, encoding the date and machine used, for instance), identifying likely paths associated with each stage of the pipeline.

Exercise 4
Display this object, and query it for the paths were Gerald analysis results are stored.

```
> sp

class: SolexaPath
experimentPath: extdata/ELAND/080828_HWI-EAS88_0003
dataPath: Data
scanPath: NA
imageAnalysisPath: C1-72_Firecrest...
baseCallPath: Bustard1.9.2_06...
analysisPath: GERALD_06-09-20..., GERALD_08-09-20...

> analysisPath(sp)

[1] "extdata/ELAND/080828_HWI-EAS88_0003/Data/C1-72_Firecrest1.9.2_06-09-2008_solexa/Bustard
[2] "extdata/ELAND/080828_HWI-EAS88_0003/Data/C1-72_Firecrest1.9.2_06-09-2008_solexa/Bustard
```

There are two analysis paths, because the data were generated by two separate runs of the ELAND *software. The analysis paths are nested inside* `baseCallPath(sp)` *as a Solexa convention, to indicate that the Gerald analyses both use the results of the same application of the base calling software.*

Note how the display of `sp` is compact, and the names in the display hint at how to navigate the object. Short read objects can be very large. Most objects in ShortRead and related packages have been designed to display a summary, rather than the 'big' data. Accessing components of objects, such as with `analysisPath` in the example above, often returns the data in all its glory – frequently, you'll want to adopt a strategy of assigning such big data to a variable, and inspecting it with functions like `str` or `head`.

A key functionality provided by ShortRead is input of a diversity of file types, both of files from the Solexa 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, e.g., to read all files containing image intensity produced by Firecrest. 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 files you are interested in.

Exercise 5
Use `list.files` *to display all files in the first Gerald analysis directory of the* `sp` *object.*

```
> list.files(analysisPath(sp)[[1]])

[1] "s_1_1_export.txt"          "s_1_1_export_head.txt"
[3] "s_1_1_sequence_head.txt" "s_1_2_export.txt"
[5] "s_2_1_export.txt"
```

A full Gerald data set would contain hundreds of files. We'll select just one file to use in subsequent analysis, based on files matching the regular expression `.*_export.txt`. These files are produced using ELAND software run in `eland-extended` mode. This mode produces files, one for each lane (and 'end' of paired end reads) that summarize diverse features of *all* reads, and is a very convenient starting point for analysis.

Exercise 6
We'll use an abbreviated file for most parts of this lab. The abbreviated file contains the first 500,000 reads from a single lane of a Solexa paired-end run. It is from lane 1, will use the first end only. The name of the file is `s_1_1_export_head.txt`*. We will use this as the 'pattern' to match, and check that we've specified the pattern appropriately*

```
> pattern <- "s_1_1_export_head.txt"
> list.files(analysisPath(sp), pattern)

[1] "s_1_1_export_head.txt"
```

Our success shouldn't be too surprising in this case, but it often pays to check. For instance, the pattern above also matches a file with the same name but with `.tar.gz` *appended!*

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. The default for this argument reads all files. An optional third argument allows the user to specify which type of file is to be read in.

Exercise 7
Use `readAligned` *to read in our abbreviated version of lane 1.* `readAligned` *is smart enough to know where alignemnt files are located in the Solexa file hierarchy.*

```
> aln <- readAligned(sp, pattern)
```

The `sp` *argument could have been replaced by a directory path, e.g.,* `"."` *(if the file were in the current working directory) or* `analysisPath(sp)`*.*

The third argument (unspecified in the above) allows input of diverse Solexa alignment files, in addition to input of MAQ text and binary alignment files and, in the development version of ShortRead, Bowtie. See the help page for `readAligned` *for additional details.*

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 8
Display the object we've read in.

```
> aln

class: AlignedRead
length: 500000 reads; width: 35 cycles
chromosome: 255:255:255 255:255:255 ... QC QC
position: NA NA ... NA NA
strand: NA NA ... NA NA
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

There are 500000 reads in the object, each read consisting of 35 nucleotides. View the first several reads and query information about, e.g., the number of reads that align to each strand, or the number of positions recorded as `NA`*.*

```
> head(sread(aln))

  A DNAStringSet instance of length 6
    width seq
[1]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[2]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[3]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[4]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[5]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[6]    35 GCAAGTTAAGAAGAGAGCAGAGAAGAACGTTTTTA

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

     -      +      *   <NA>
119313 119685      0 261002

> sum(is.na(position(aln)))

[1] 261002
```

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.*

What are all the `NA` values returned by `strand` and `position`? These correspond to reads that did not align to the reference genome used by ELAND; that about 1/2 the reads do not align is below normal, making this an interesting opportunity for quality assessment. 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.

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 ELAND alignment; details of calibration are to be found in Illumina documentation. The alignments themselves also have qualities associated with them, with the details of alignment quality differing between alignment algorithms.

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

```
> head(quality(aln))
```

```
class: SFastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[2]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[3]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[4]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[5]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[6]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
```

These qualities are string-encoded $-10\log_{10}$ probabilities. The encoding in this case follows a convention established by Solexa. 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. These are easily manipulated, 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] 21.45995 19.91676 17.98661 19.67178 18.27315 18.61814
 [7] 18.99522 18.81180 18.69941 18.47509 19.12563 17.39689
[13] 18.26270 18.85640 17.82326 17.43106 17.95288 17.01137
[19] 17.56018 17.73439 16.51199 16.99086 17.80355 16.38983
[25] 17.35305 17.15449 15.99980 16.38890 16.59765 16.18294
[31] 13.48905 13.03266 13.00785 12.75729 11.96461
```

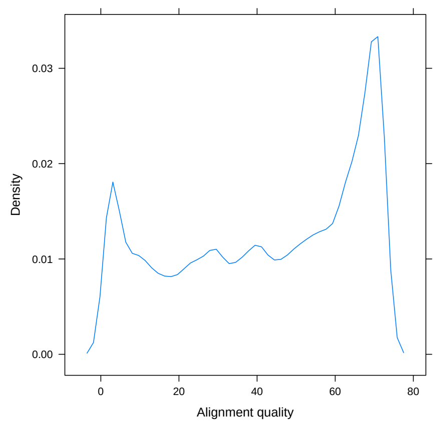

Figure 1: Alignment quality

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 (in this run of ELAND, detailed alignment information is provided only for reads that align to unique locations in the genome).

Exercise 10
Retrieve the alignment quality scores, determine how many align poorly, and visualize the distribution (figure 1) of scores.

```
> alignQuality(aln)
```

```
class: NumericQuality
quality: 0 0 ... 0 0 (500000 total)
```

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

```
[1] 295387
```

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

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 this data and a metadata description of it. For instance, ELAND includes information about whether the read passed a base-calling filter (based on strength and consistency of early bases in the read), in addition to the lane, tile, x and y coordinate of each read.

Exercise 11
Use the `alignData` *function to extract the additional information in the* ELAND *alignment file. The underlying data in this object can be accessed as though it were a data frame, for instance to tally the number of reads passing Solexa base calling filter.*

```
> alignData(aln)

An object of class "AlignedDataFrame"
  readName: 1, 2, ..., 500000  (500000 total)
  varLabels and varMetadata description:
    run: Analysis pipeline run
    lane: Flow cell lane
    ...: ...
    filtering: Read successfully passed filtering?
    (6 total)

> table(alignData(aln)$filtering)

     Y      N
287222 212778
```

2.3 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 passing Solexa filtering, and aligning to the reference genome.

```
> filtIdx <- alignData(aln)$filtering == "Y"
> alignedIdx <- !is.na(strand(aln))
> aln[filtIdx & alignedIdx]

class: AlignedRead
length: 197432 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

A different approach to subsetting is to use objects of class *SRFilter*. These can be particularly useful as an argument to `readAligned`, in addition to use in interactive sessions.

Exercise 13
Construct instances of built-in filters to select reads passing the Solexa filtering criterion, and uniquely aligning to a fully assembled chromosomes. These can be 'composed' into a single overall filter, and applied to restrict available reads.

```
> filt1 <- alignDataFilter(expression(filtering ==
+     "Y"))
> filt2 <- chromosomeFilter("chr[0-9XYM]+.fa")
> filt <- compose(filt1, filt2)
> caln <- aln[filt(aln)]
> caln

class: AlignedRead
length: 195719 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

The filters developed above could be used to filter reads while being read in to R, e.g,. with

```
> readAligned(sp, pattern, filter = filt)
```

The `srFilter` function can be used to create custom filters. The idea is that filter functions accept a single argument `x` that is an object to be filtered, and returns a logical vector that can be used to select elements of the object.

Exercise 14
As a first example, write and use a filter to select only a single read from all that align to a particular chromosome, position, and strand.

```
> ualignFilter <- srFilter(function(x) {
+     ## create a numerical index of reads. Divide the index, position,
+     ## and strand information between chromosomes. Select the index of
+     ## a single read at each unique position and strand. Return the
+     ## selected index as a logical vector with the same length as x
+     oindex <- seq_len(length(x))
+     index <- tapply(oindex, chromosome(x), c)
+     pdup <- tapply(position(x), chromosome(x), duplicated)
+     sdup <- tapply(strand(x), chromosome(x), duplicated)
+     keep <- oindex  %in% unlist(mapply(function(i, p, s) {
+         i[!(p & s)]
+     }, index, pdup, sdup))
+ }, name="select only one read per position & strand ")
> caln[ualignFilter(caln)]
```

```
class: AlignedRead
length: 188219 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

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("chr2.fa")` creates an instance of the chromosome filter to select only chromosomes matching `chr2.fa`.

Exercise 15
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 = "Martin's demo sampling filter")
+ }
> sample100 <- samplingFilter(100)
> caln[sample100(caln)]
```

```
class: AlignedRead
length: 100 reads; width: 35 cycles
chromosome: chr4.fa chr17.fa ... chr13.fa chr2.fa
position: 152844856 35197700 ... 86362072 163646486
strand: + - ... - +
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

2.4 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., 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.
- Character-encoded base quality scores are interpreted as the default for the software package being parsed, e.g., as 'Solexa fastq' for ELAND. The object returned by `quality` applied to an *AlignedRead* object is either *FastqQuality* or *SFastqQuality*.

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

3 Additional input functions

ShortRead, Biostrings, and the standard input functions from R provide additional tools for reading Solexa and other alignment formats. The `readXStringColumns` function provides a convenient way to read DNA and quality sequences in to compact data structures. `readFasta` and its counterpart `readFastq` provide tools for reading fastq- or fastq- (i.e., including quality annotation) formatted files.

Exercise 16
Files `_sequence.txt` contain fastq-formatted sequence and quality scores. A sample of this file type is available. Read these in to data structured defined in ShortRead. The content of `reads` can be retrieved with functions `id`, `sread`, and `quality`.

```
> ap <- analysisPath(sp)[[1]]
> reads <- readFastq(ap, "s_1_1_sequence_head.txt$")
> head(id(reads))

  A BStringSet instance of length 6
    width seq
[1]    24 HWI-EAS88_3:1:1:33:484/1
[2]    24 HWI-EAS88_3:1:1:33:272/1
[3]    24 HWI-EAS88_3:1:1:31:594/1
[4]    24 HWI-EAS88_3:1:1:33:383/1
```

```
[5]     24 HWI-EAS88_3:1:1:35:216/1
[6]     25 HWI-EAS88_3:1:1:883:458/1
```

The `readPrb` functions reads 'raw' base call quality scores from `_prb` files in the `baseCallPath` directory; the result is a *BStringSet* object that compactly represents the quality scores in a way analogous to the results of `quality` applied to `aln`.

Exercise 17
The `_export.txt` files read by `readAligned` are tab-delimited text files. The goal of this exercise is to read the DNA sequence and quality score columns in to R as DNAStringSet and BStringSet objects DNAStringSet and BStringSet contain classes the represent DNA or 'biological' strings; the content extends from the base class XString.

Start by parsing the first line of a `_prb` file to get a sense of its content. Specify the `colClasses` to be imported, using `NULL` to indicate that a column should be skipped, and `DNAString` or `BString` to indicate columns that are to be read as corresponding sets. Then read the file with `readXStringColumns`.

```
> fl <- list.files(ap, pattern, full = TRUE)
> cols <- strsplit(readLines(fl, 1), "\t")[[1]]
> length(cols)

[1] 22

> cols[9:10]

[1] "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
[2] "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU"

> colClasses <- rep(list(NULL), 22)
> colClasses[9:10] <- c("DNAString", "BString")
> strings <- readXStringColumns(ap, pattern, colClasses = colClasses)
> head(strings[[2]])

  A BStringSet instance of length 6
    width seq
[1]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[2]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[3]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[4]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[5]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[6]     35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
```

For each column in `colClasses`, `readXStringColumns` parses the corresponding column in all files matching the function argument `pattern` into a single XStringSet, i.e., concatenating the content of all files into a single object.

Many of the files produced by Solexa are simple text files. These can be read in using standard R input commands.

Exercise 18
The `_int` files in the `imagaAnalysisPath` are tab- and space-delimited records of intensities measured along cycles of reads. Each line corresponds to a single cluster (i.e., putative read). The first four numbers indicate the lane, tile, x, and y. Subsequent numbers come in groups of 4, corresponding the intensities of the A, C, G, and T nucleotides over successive cycles. Here are coordinates of the first read, followed by the first 3 sets of intensities.

```
> fl <- list.files(imageAnalysisPath(sp), ".*_int.*",
+     full = TRUE)
> strsplit(readLines(gzfile(fl, open = "rb"), 1),
+     "\t")[[1]][1:7]
```

```
[1] "1"
[2] "1"
[3] "33"
[4] "484"
[5] "5571.5   40.1  153.0   12.9"
[6] "5017.6   22.9  141.5    7.5"
[7] "4565.6   56.2  159.9   16.2"
```

The files are compressed, so we wrap the file in a `gzfile` function call to unzip the file prior to parsing. We now read the files into an object, and then convert the intensities into a standard array:

```
> int <- readIntensities(sp)
> arr <- as(intensity(int), "array")
```

Perhaps surprisingly, the intensities for the four nucleotides are not independent of one another (figure 2). At later cycles, the amplitude of the intensities shrink toward zero and become less orthogonal. This presumably contributes to decreased quality of base calls.

```
> print(splom(arr[, , 5], pch = ".", log = "xy"))
```

4 Quality assessment

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

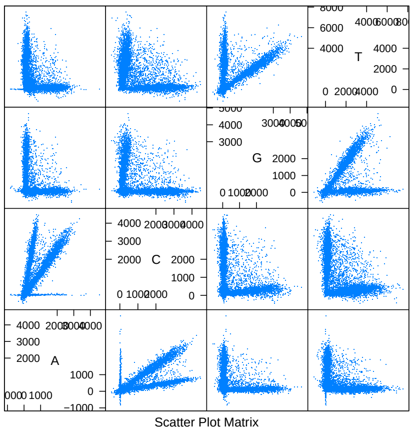

Figure 2: Intensities from cycle 5

4.1 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 `_export.txt` file, and extracts information on reads and their qualities. Evaluation of the function is straight-forward, e.g., `qa <- qa(sp)`, to visit all files in the `sp` *SolexaPath*. 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 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 19
Rather than collating information during the lab, we load the data from a previously stored instance.

```
> load(file.path("data", "qa_080828_081110.rda"))
> qa

class: SolexaExportQA(9)
QA elements (access with qa[["elt"]]):
  readCounts: data.frame(8 3)
  baseCalls: data.frame(8 5)
  readQualityScore: data.frame(12288 4)
  baseQuality: data.frame(752 3)
  alignQuality: data.frame(629 3)
  frequentSequences: data.frame(1200 4)
  sequenceDistribution: data.frame(4540 4)
  perCycle: list(2)
    baseCall: data.frame(1400 4)
    quality: data.frame(6664 5)
  perTile: list(2)
    readCounts: data.frame(7200 4)
    medianReadQualityScore: data.frame(7200 4)
```

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

The QA information collated from the `_export.txt` files is summarized into a PDF report using the Rfunctionreport function. This function currently requires a LaTeX installation, although the intention is that the report will eventually be generated in different formats. The command to create the report is `rpt <- report(qa, dest=tempfile())`. This creates a PDF file at the location specified by the argument `dest`.

Exercise 20
Rather than create a report, we provide a sample, derived from the `qa` *object loaded in the previous exercise. The sample is at docs/qa_080828_081110.pdf.*

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.

4.2 Exploring qa

The `qa` object is a list-like structure with several entities.

Exercise 21
The `readCounts` *element of* `qa` *is a simple data frame summarizing, on a per-lane bases, the total number of reads, the number reads passing Solexa internal filtering, and the number of aligned reads.*

```
> qa[["readCounts"]]

                     read filtered aligned
s_1_1_export.txt 3668433  2278945 1910666
s_2_1_export.txt 4230424  3239956 2771169
s_3_1_export.txt 4003465  3089375 1720396
s_4_1_export.txt 4521919  3446177 2571235
s_6_1_export.txt 4004807  3127297 1985855
s_7_1_export.txt 3546869  2732974 1368590
```

```
s_8_1_export.txt 4232977  3291627 2379709
s_5_1_export.txt 2842633  2399387 2320140
```

Lanes 1-4 and 6-8 correspond to biologically interesting samples; lane 5 is the Solexa φX-174 control lane. The number of reads (between 2.8 and 4.5 million) is low for a typical experiment (official guidelines are provided in Solexa documentation).

It can be difficult to scan large numbers, so the QA report template defines functions that help to display the information in a more comprehensible fashion. Source these files into your current R session, and view the second and third columns a proportion of the first.

```
> source(file.path("scripts", "qa_solexa.R"))
> ppnCount(qa[["readCounts"]])

                    read  filtered   aligned
s_1_1_export.txt 3668433 0.6212312 0.5208398
s_2_1_export.txt 4230424 0.7658703 0.6550570
s_3_1_export.txt 4003465 0.7716753 0.4297267
s_4_1_export.txt 4521919 0.7621050 0.5686159
s_6_1_export.txt 4004807 0.7808858 0.4958678
s_7_1_export.txt 3546869 0.7705314 0.3858586
s_8_1_export.txt 4232977 0.7776151 0.5621833
s_5_1_export.txt 2842633 0.8440720 0.8161940
```

Refer to the QA report for further commentary on this and other aspects of the report

4.3 Frequent sequences

A feature of raw reads, and of many subsequent stages of short read analysis, is a power law-like relationship between the number of times a read occurs, and the number of occurrences of a particular sequence in the sample. This information is contained in the `sequenceDistribution` element of `qa`, and is the result of running the `tables` command (defined in ShortRead) on a *DNAStringSet* object.

Exercise 22
Retrieve the `sequenceDistribution` *element from* `qa`*; it is a simple data frame. Look at the contents of the data frame using* `head`*, and select just lane 5 raw reads. plot the power-law relationship between the number of reads and number of times reads occur. As an alternative display of the same information, plot the cumulative number of reads as a function of the number of times a read occurs.*

```
> df <- qa[["sequenceDistribution"]]
> df5raw <- df[df$lane == "s_5_1_export.txt" & df$type ==
+     "read", ]
> head(df5raw)
```

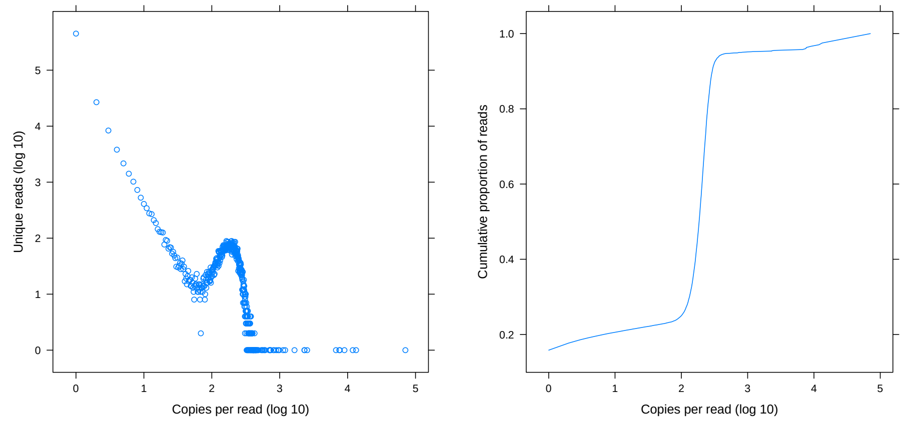

Figure 3: Number of copies of unique reads

```
     nOccurrences nReads type             lane
3337            1 450124 read s_5_1_export.txt
3338            2  26810 read s_5_1_export.txt
3339            3   8366 read s_5_1_export.txt
3340            4   3806 read s_5_1_export.txt
3341            5   2166 read s_5_1_export.txt
3342            6   1417 read s_5_1_export.txt
```

```
> print(xyplot(log10(nReads) ~ log10(nOccurrences),
+     df5raw, xlab = "Copies per read (log 10)",
+     ylab = "Unique reads (log 10)"))
```

```
> csum <- with(df5raw, cumsum(nReads * nOccurrences))
> csum <- csum/csum[length(csum)]
> print(xyplot(csum ~ log10(nOccurrences), df5raw,
+     xlab = "Copies per read (log 10)", ylab = "Cumulative proportion of reads",
+     type = "l"))
```

Results appear in figure 3. The cumulative form of this figure appears in the QA report.

The power-law relationship between copies per read and number of unique reads in the control lane consists of three components. At the left of the graph are $> 10^5$ reads that are each represented by only one copy. These likely correspond to sequencing, base calling, or other errors associated with the technology. At the right of the figure are a small number of reads represented many times. These 'frequent' sequences are summarized in the `frequentSequences` element of `qa`; frequent sequences are also reported by the `tables` function of ShortRead.

Exercise 23
Discover the frequent sequences amongst the raw and aligned reads of lane 5.

```
> df <- qa[["frequentSequences"]]
> head(df[df$lane == "s_5_1_export.txt" & df$type ==
+     "read", 1:2])

                                 sequence count
1051 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 70947
1052 ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 13320
1053 TNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 11892
1054 CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  8978
1055 GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  7670
1056 GATCTTTGGCGGCACGGAGCCGCGCATCACCTGTA  7561
```

Frequent sequences include poly-A reads, reads where only a few bases were called, and reads with close similarity to the Solexa primer or adapter sequence used in sample preparation.

Exercise 24
Many of the primer sequences are filtered out by Solexa criteria, but it is worth discovering how many reads are 'similar' to this sequence. Use the `srdistance` *function to identify such reads, e.g., amongst those reads that contain no* `N` *nucleotides.*

```
> seq <- "CGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGT"
> dist <- srdistance(clean(aln), seq)[[1]]
> head(table(dist))

dist
  0   1   2   3   4   5
218 107  59  31  25  17
```

`srdistance` returns the edit distance between each read and the reference sequence, where the edit distance is defines so that each base mismatch represents an additional increment of 1. There are 78 reads that differ at 2 or fewer locations, from amongst the 370049 reads in the cleaned sample used in this exercise.

Reads represented many times may be problematic for downstream analysis. For instance, sample preparation protocols may involve a PCR step that results in differential amplification, whereas the analysis assumes reads are represented in proportion to their occurrence. The `ualignFilter` function defined above, and the `srduplicated` function in ShortRead represent two approaches to dealing with this problem by ensuring that reads are represented exactly once (by some definition of 'once'!).

Sequences in the middle portion of the graph in figure 3 will often, depending on the nature of the investigation, represent the sequences of main biological interest. These are sequences represented an intermediate number of times, as might be required for reasonable coverage in a SNP discovery or ChIP-seq experiment. The right-hand graph in figure 3 shows a relatively abrupt transition between reads represented rarely and those represented many times.

The sample QA report shows that the non-control lanes show much broader transitions from rarely to frequently represented reads. This could represent technical shortcomings of this run (e.g., inadequate enrichment of sample DNA) or features of intrinsic biological interest (e.g., wide variability in ChIP abundance between binding sites). Regardless of ultimate source, the broad distribution of read occurrences implies significant effort may be required to distinguish 'noise' (reads corresponding to those in the left and right portions of the control lane graph) from signal.

Finally, while the control lane shows a relatively abrupt transition between reads that occur rarely and those that are common (figure 3), the distribution is in fact 3- or 4-fold broader than expected under a naive model of random read starts along the φX-174 genome. This is reinforced by alignments to the reference genome, where clear patterns (e.g., unequal representation on plus and minus strands; non-uniform coverage) are apparent.

Exercise 25
As an advanced exercise, simulate reads selected uniformly along both strands of the 5200bp long φX-174 genome. Compare the times each read is represented in your sample with those from the actual control lane. Hint: use `sample` with `replace=TRUE` to generate the reads, and `table(table(reads))` to summarize their occurrence; this should take less than 5 lines of R code.

4.4 Cycle-specific qualities and base calls

As a final foray into the details of quality assessment, consider base calls and quality, and how these change across cycles (see the second table and section 4 of the QA report).

The table in the QA report suggests that the control lane (lane 5) is enriched for A and T; this is confirmed by the figure in section 4. This likely reflects underlying differences in the genomic regions represented in each lane.

Exercise 26
Use `alphabetFrequency` to summarize nucleotide use in the short reads in `aln`, from the first part of this lab.

```
> alphabetFrequency(sread(aln), collapse = TRUE,
+     baseOnly = TRUE, freq = TRUE)

         A          C          G          T      other
0.25588029 0.22092086 0.20225720 0.22800937 0.09293229
```

The frequency of 'other' (i.e., uncalled) nucleotides ($> 9\%$) is very high; typical runs are $< 3\%$; recent runs with GAII technologies after 36 cycles are $< 1\%$.

An unexpected aspect of the figure in section 4 of the QA report is apparent trends in nucleotide frequency with cycle. For instance, all lanes show a marked decrease in A and increase in C across cycles. This is unexpected in this experiment, where the *a priori* expectation is that sequences start at essentially arbitrary locations in the sequenced DNA: an A is expected as frequently at the beginning of the sequence as at the end.

Exercise 27
Use `alphabetByCycle` *to extract the number of each nucleotide sequenced at each cycle. Convert the matrix into a data frame containing only the called nucleotides, and plot these counts as a function of cycle.*

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

```
[1] 17 35
```

```
> abc[1:4, 1:5]
```

```
         cycle
alphabet    [,1]    [,2]    [,3]    [,4]    [,5]
       A  145955  147044  128640  140609  135751
       C  117538  106357   98356  109948  101961
       G  118619  109879  100303  112267  102233
       T  115666  118470  106223  112917  107716
```

```
> abc <- abc[rowSums(abc) != 0, ]
> df <- as.data.frame(t(abc[1:4, ]))
> print(xyplot(A + C + G + T ~ 1:nrow(df), df, type = "l",
+     auto.key = list(x = 0.75, y = 0.95, points = FALSE,
+         lines = TRUE), xlab = "Cycle", ylab = "Count"))
```

There are a number of possible contributors to cycle-dependent nucleotide frequencies, including inadequate reagent volume, and nucleotide-specific differential accumulation of fluorescent dyes. Figure 4 and the figure in section 4 of the PDF contain additional features that are moderately unexpected, and unexplained. For instance, the frequency of a nucleotide such as A changes very systematically across cycles, first increasing and then decreasing; this seems more regular than expected. Patterns of nucleotide change also seem to echo one another at similar cycles but in different lanes, even when the lanes have different biological material. This occurs for instance in lanes 1-4 of the QA report, where the last 5 cycles of the C nucleotide seem to change in (comparative!) unison.

5 An advanced note

Several functions are designed to use the Rmpi package for distributed computation, if installed. For instance

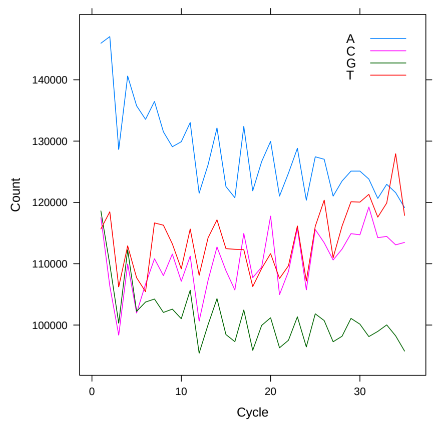

Figure 4: Nucleotide frequency per cycle, lane 1

```
> library(Rmpi)
> mpi.spawn.Rslaves(nsl = 8)
> qa <- qa(sp)
> mpi.close.Rslaves()
```

distributes the calculation of quality assurance summaries across 8 processors. The `srapply` function can be used to distribute your own calculations.

6 Summary

This portion of the course provides an overview of the input and quality assessment functionality available in the ShortRead package. A summary of the insights learned might be reflected in simple, and somewhat naive, work flows.

The first work flow simply performs quality assessment on ELAND aligned data.

```
> sp <- SolexaPath("extdata/ELAND/080828_HWI-EAS88_0003")
> rpt <- report(qa(sp), dest = "reports/my_report.pdf")
```

Performing QA on ELAND data does not commit us to using ELAND alignments in subsequent steps.

The second work flow reads aligned data in to R. The work flow might start with aligned reads created by one of many aligners; we start with ELAND `_export.txt` files. There are many possible issues highlighted in the forgoing discussion. We choose to establish a series of filters to eliminate some reads at the very start of our work flow. We eliminate reads with ambiguous base calls and failing Solexa's internal filtering criteria. Reads aligning to multiple locations in the genome are not straight-forward to deal with, and are not essential for ChIP-seq style experiments (this is not the case for expression or RNA-seq experiments), so we remove these. Most close matches to the Solexa primer sequence are flagged as not passing Solexa base call filters, but we eliminate reads near to this as well. Finally, we restrict our attention to those reads that align to assembled chromosomes, putting aside for the moment those reads aligning to organelle genomes (the X and Y chromosomes also require special consideration, and we might often eliminate these from our initial work flow, too). Our second work flow is thus:

```
> filt1 <- nFilter()
> filt2 <- alignDataFilter(expression(filtering=="Y"))
> filt3 <- alignQualityFilter(threshold=1)
> filt4 <- srdistanceFilter("CGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGT", 4)
> filt5 <- chromosomeFilter("chr[0-9XY]+.fa")
> filt <- compose(filt1, filt2, filt3, filt4, filt5)
> aln <- readAligned(sp, "s_1_1_export.txt$", filter=filt)
```

This work flow applies equally to MAQ aligned data, with the exception that Solexa filtering criteria are not available and the chromosome naming convention in the MAQ-aligned reads in our sample are different:

```
> maqDir <- file.path("extdata", "MAQ")
> filt5 <- chromosomeFilter("chr[0-9XY]+$")
> filt <- compose(filt1, filt3, filt4, filt5)
> maq <- readAligned(maqDir, "s_8.map", "MAQMap",
+     filter = filt)
```

Each of the filters represents a decision. The decision may be inappropriate for particular analyses, and may be revisited as understanding of the data matures; indeed, analyses in other portions of this course are based on different subsets of the data!

7 Session information

```
> sessionInfo()

R version 2.9.0 Under development (unstable) (2009-01-12 r47568)
i686-pc-linux-gnu

locale:
LC_CTYPE=C;LC_NUMERIC=C;LC_TIME=C;LC_COLLATE=C;LC_MONETARY=C;LC_MESSAGES=en_US.UTF-8;LC_PAPE

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

other attached packages:
```

```
[1] ShortRead_1.1.36   lattice_0.17-20    BSgenome_1.11.9
[4] Biobase_2.3.9      Biostrings_2.11.25 IRanges_1.1.34

loaded via a namespace (and not attached):
[1] Matrix_0.999375-18 grid_2.9.0
```