An Introduction to *ShortRead*

Martin Morgan

Modified: 28 September 2010. Compiled: January 5, 2012

```
> library("ShortRead")
```

The *ShortRead* package aims to provide key functionality for input, quality assurance, and basic manipulation of 'short read' DNA sequences such as those produced by Solexa, 454, and related technologies, including flexible import of common short read data formats. This vignette introduces key functionality.

Support is most fully developed for Solexa; contributions from the community are welcome.

1 A first workflow

This section walks through a simple work flow. It outlines the hierarchy of files produced by Solexa. It then illustrates a common way for reading short read data into R.

1.1 *SolexaPath*: navigating Solexa output

SolexaPath provides functionality to navigate files produced by Solexa Genome Analyzer pipeline software. A typical way to start a *ShortRead* session is to point to the root of the output file hierarchy. The *ShortRead* package includes a very small subset of files emulating this hierarchy. The root is found at

```
> exptPath <- system.file("extdata", package="ShortRead")
```

Usually `exptPath` would be a location on the users' file system. Key components of the hierarchy are parsed into R with

```
> sp <- SolexaPath(exptPath)
> sp
```

```
class: SolexaPath
experimentPath: /tmp/RtmpHztr4P/Rinst43fd78cb6e6/ShortRead/extdata
dataPath: Data
scanPath: NA
imageAnalysisPath: C1-36Firecrest
baseCallPath: Bustard
analysisPath: GERALD
```

`SolexaPath` scans the directory hierarchy to identifying useful directories. For instance, image intensity files are in the 'Firecrest' directory, while summary and alignment files are in the analysis directory

```
> imageAnalysisPath(sp)
```

```
[1] "/tmp/RtmpHztr4P/Rinst43fd78cb6e6/ShortRead/extdata/Data/C1-36Firecrest"
```

```
> analysisPath(sp)
```

```
[1] "/tmp/RtmpHztr4P/Rinst43fd78cb6e6/ShortRead/extdata/Data/C1-36Firecrest/Bustard/GERALD"
```

Most functionality in *ShortRead* uses `baseCallPath` or `analysisPath`. Solexa documentation provides details of file content. `SolexaPath` accepts additional arguments that allow individual file paths to be specified.

Many functions for Solexa data input 'know' where appropriate files are located. Specifying `sp` is often sufficient for identifying the desired directory path. Examples of this are illustrated below, with for instance `readAligned` and `readFastq`.

Displaying an object, e.g., `sp`, provides hints at how to access information in the object, e.g., `analysisPath`. This is a convention in *ShortRead*.

1.2 `readAligned`: reading aligned data into R

Solexa `s_N_export.txt` files (`_N_` is a placeholder for the lane identifier) represent one place to start working the short read data in R. These files result from running ANALYSIS eland_extended in the Solexa Genome Analyzer. The files contain information on all reads, including alignment information for those reads successfully aligned to the genome. *ShortRead* parses additional alignment files, including MAQ binary and text (`mapview`) files and Bowtie text files; consult the help page for `readAligned` for details. *ShortRead* flexibly parses many other Solexa files; aligned reads represent just one entry point.

To read a single `s_N_export.txt` file into R, for instance from lane 2, use the command

```
> aln <- readAligned(sp, "s_2_export.txt")
> aln
```

```
class: AlignedRead
length: 1000 reads; width: 35 cycles
chromosome: NM NM ... chr5.fa 29:255:255
position: NA NA ... 71805980 NA
strand: NA NA ... + NA
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

This illustrates the convention used for identifying files for input into R and used by *ShortRead*. The function takes a directory path and a pattern (as a regular

expression, similar to the R function `list.files`) of file names to match in the directory. Usually, all files matching the pattern are read into a *single* R object; this behavior is desirable for several of the input functions in *ShortRead*. In the present case the usual expectation is that a single `s_N_export.txt` file will be read into a single R object, so the `pattern` argument will identify a single file.

1.2.1 Input of other aligned read files

ELAND software provides access to much interesting data, in addition to alignments, but if the interest is in aligned reads then input may come from any of a number of different software packages. Many of these alignments can be input with *ShortRead*.

Bowtie is a very fast aligner, taking a few tens of minutes to align entire lanes of reads to reference genomes. Use `readAligned` with the `type="Bowtie"` argument to input alignments. Reading Bowtie output using `readAligned` produces the same class of object as reading ELAND output. Like ELAND, Bowtie provides information on short read, quality, chromosome, position, and strand; there is no information on alignment quality avaiable from Bowtie. ELAND and Bowtie provide very different auxiliary information. Consult the `readAligned` and help page Bowtie manual for additional detail.

MAQ is another poplar aligner. *ShortRead* can input MAQ binary or text formats (see the arguments `type="MAQMapShort"`, `"MAQMap"`, and `"MAQMapview"`). As with Bowtie, MAQ provides essential information about reads and their aligments, plus additional information that differs somewhat from the additional information provided by ELAND.

Alignment information may come in a variety of different text-based formats. Not all of these will be supported by *ShortRead*. There are a number of tools available to input this into R.

A basic strategy is involves two passes over the data, followed by synthesis of results into an *AlignedRead* object. First, input alignment data using functions such as `read.table`. Use the `colClasses` argument to 'mask-out' (i.e., avoid importing) DNA and quality sequences. Next, use `readXStringColumns` or `readFastq` to import the short read and quality information. Finally, use the alignment data and reads as arugments to the `AlignedRead` function to synthesize the input. The following illustrates use of `readXStringColumns` and `readFastq`. These functions receive further attention below.

1.2.2 Cautions

There are several confusing areas of input. (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).
- 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 intrepreted 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 custommize output; such customization needs to be accomodated when reads are input using *ShortRead*.

1.2.3 Filtering input

Downstream analysis may often want to use a well-defined subset of reads. These can be selected with the `filter` argument of `readAligned`. There are built-in filters, for instance to remove all reads containing an `N` nucleotide, to select just those reads that map to the genome file `chr5.fa`, to select reads on the `+` strand, or to 'level the playing field' by selecting only a single read for any chromosome, position and strand:

```
> nfilt <- nFilter()
> cfilt <- chromosomeFilter('chr5.fa')
> sfilt <- strandFilter("+")
> ofilt <- occurrenceFilter(withSread=FALSE)
```

Here we select only those reads that map to `chr5.fa`:

```
> chr5 <- readAligned(sp, "s_2_export.txt", filter=cfilt)
```

Filters can be 'composed' to act in unison, e.g., selecting only reads mapping to `chr5.fa` and on the `+` strand:

```
> filt <- compose(cfilt, sfilt)
> chr5plus <- readAligned(sp, "s_2_export.txt", filter=filt)
```

Filters can subset aligned reads at other stages in the work flow, using a paradigm like the following:

```
> chr5 <- aln[cfilt(aln)]
```

Users can easily create their own filter by writing a function that accepts an object of class `AlignedRead`, and returns a logical vector indicating which reads in the object pass the filter. See the example on the `srFilter` help page for details, and for information about additional built-in filters.

1.3 Exploring *ShortRead* objects

`aln` is an object of *AlignedRead* class. It contains short reads and their (calibrated) qualities:

```
> sread(aln)

  A DNAStringSet instance of length 1000
       width seq
   [1]    35 CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC
   [2]    35 AGCCTCCCTCTTTCTGAATATACGGCAGAGCTGTT
   [3]    35 ACCAAAAACACCACATACACGAGCAACACACGTAC
   [4]    35 AATCGGAAGAGCTCGTATGCCGGCTTCTGCTTGGA
   [5]    35 AAAGATAAACTCTAGGCCACCTCCTCCTTCTTCTA
   [6]    35 AAAAAAAAAAAGGACACACCATGAGATCACAGGGA
   [7]    35 TAAAAAATTAGCAAAAAACCAAAAATGTAATTGAT
   [8]    35 TAAATCGTGCTGTAACCTTTCCCAACATCTCTGTG
   [9]    35 AATGACCGATAATTAAAAATAAAATCTTTGCATAT
   ...   ... ...
 [992]    35 GAAAAAAAAACAGAACGATGCGTTCATCCACGGCA
 [993]    35 TTATCCCTGGTTTCTCCTTGTGACTCTCTGTTGTC
 [994]    35 AGAGCTTTAGGCAGCTCGGTGTGTCCTTTCTATTC
 [995]    35 TATATTGCCCCCTGCAGCAATGCCCCTTACCCGTC
 [996]    35 GTGGCAGCGGTGAGGCGGCGGGGGGGGGTTGTTTG
 [997]    35 GTCGGAGGTCAGCAAGCTGTAGTCGGTGTAAAGCT
 [998]    35 GTCATAAATTGGACAGTGTGGCTCCAGTATTCTCA
 [999]    35 ATCTACATTAAGGTCAATTACAATGATAAATAAAA
[1000]    35 TTCTCAGCCATTCAGTATTCCTCAGGTGAAAATTC

> quality(aln)

class: SFastqQuality
quality:
  A BStringSet instance of length 1000
       width seq
   [1]    35 YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH
   [2]    35 ZXZUYXZQYYXUZXYZYYZZXXZZIMFHXQSUPPO
   [3]    35 LGDHLILLLLLLLLIGFLLALDIFDILLHFIAECAE
   [4]    35 JJYYIYVSYYYYYYYYSDYYWVUYYNNVSVQQELQ
   [5]    35 LLLILIIIDLLHLLLLLLLLLLALLLLHLLLLEL
```

```
   [6]    35 YYYYYYYWVVMGGUHQHQMUFMICDMCDHQHEDDD
   [7]    35 ZZZZZZZZZZYZZZZZYZZZZYYZZZZZZZUUUUU
   [8]    35 ZZZZZZZZUZZUZZZZZZZZZZZZYZYZZZUUHUH
   [9]    35 ZZZZZZZYZZYZZZZYZZZZZZZZZZZZZXUNUUU
   ...   ... ...
 [992]    35 YYYVVVSSGVSQIGIUSFFYIHLUUHFQXULPLLH
 [993]    35 ZZZZZZZZZZZZZZZZZZZYZXZZZZZZZSUUJUU
 [994]    35 YIOSMSGSYOSUIYUSUDLIWUQIQQUUUFPLENG
 [995]    35 ZZZZZZZZZZZZZZYZXYZZZXZZZXZZZSZUUUUU
 [996]    35 ZZZZZZZYZZYUYZYUYZKYUDUZIYYODJGUGAA
 [997]    35 ZZZZZZZZZZZZZZZZZZYZZYXXZYSSXXUUHHQ
 [998]    35 ZZZZZZZZZZZZZZZZYZZZZYZZZZYZZXZUUUUS
 [999]    35 ZZZZZZZZZZZZYXZYZYZZYZYZZXKZSYXUUNUN
[1000]    35 ZZZZZZZZZZZZZZZYZZZZZZZZYYSYSZXUUUUU
```

The short reads are stored as a *DNAStringSet* class. This class is defined in *Biostrings*. It represents DNA sequence data relatively efficiently. There are a number of very useful methods defined for *DNAStringSet*. Some of these methods are illustrated in this vignette. Other methods are described in the help pages and vignettes of the *Biostrings* and *IRanges* packages.

Qualities are represented as SFastqQuality-class objects. The qualities in the `aln` object returned by `readAligned` are of class *BStringSet*. The *BStringSet* class is also defined in *Biostrings*, and shares many methods with those of *DNAStringSet*.

The `aln` object contains additional information about alignments. Some of this additional information is expected from any alignment, whether generated by Solexa or other software. For example, `aln` contains the particular sequence within a target (e.g., chromosomes in a genome assembly), the position (e.g., base pair coordinate), and strand to which the alignment was made, and the quality of the alignment. The display of `aln` suggests how to access this information. For instance, the strand to which alignments are made can be extracted (as a factor with three levels and possibly `NA`; the level `"*"` corresponds to reads for which strand alignment is intrinsically not meaningful, whereas `NA` represents the traditional concept of information not available, e.g., because the read did not align at all) and tabulated using familiar R functions.

```
> whichStrand <- strand(aln)
> class(whichStrand)

[1] "factor"

> levels(whichStrand)

[1] "+" "-" "*"

> table(whichStrand, useNA="ifany")
```

```
whichStrand
   +    -    * <NA>
 203  203    0  594
```

This shows that about 59.4% of reads were not aligned (level `NA`).

The `aln` object contains information in addition to that expected of all alignments. This information is accessible using `alignData`:

```
> alignData(aln)
```

```
An object of class "AlignedDataFrame"
  readName: 1 2 ... 1000 (1000 total)
  varLabels: run lane ... contig (7 total)
  varMetadata: labelDescription
```

Users familiar with the *ExpressionSet* class in *Biobase* will recognize this as an *AnnotatedDataFrame*-like object, containing a data frame with rows for each short read. The *AlignedDataFrame* contains additional meta data about the meaning of each column. For instance, data extracted from the Solexa export file includes:

```
> varMetadata(alignData(aln))
```

```
                            labelDescription
run                      Analysis pipeline run
lane                            Flow cell lane
tile                            Flow cell tile
x                           Cluster x-coordinate
y                           Cluster y-coordinate
filtering Read successfully passed filtering?
contig                                   Contig
```

Guides to the precise meaning of this data are on the help page for the *AlignedRead* class, and in the manufacturer manuals.

Simple information about the alignments can be found by querying this object. For instance, unaligned reads have `NA` as their position, and reads passing Solexa 'filtering' (their base purity and chastity criteria) have a component of their auxiliary `alignData` set to `"Y"`. Thus the fraction of unaligned reads and reads passing filtering are

```
> mapped <- !is.na(position(aln))
> filtered <- alignData(aln)[["filtering"]] =="Y"
> sum(!mapped) / length(aln)
```

```
[1] 0.594
```

```
> sum(filtered) / length(aln)
```

```
[1] 0.764
```

Extracting the reads that passed filtering but were unmapped is accomplished with

```
> failedAlign <- aln[filtered & !mapped]
> failedAlign
```

```
class: AlignedRead
length: 400 reads; width: 35 cycles
chromosome: NM NM ... NM 29:255:255
position: NA NA ... NA NA
strand: NA NA ... NA NA
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

Alternatively, we can extract just the short reads, and select the subset of those that failed filtering.

```
> failedReads <- sread(aln)[filtered & !mapped]
```

1.4 Quality assessment

The `qa` function provides a convenient way to summarize read and alignment quality. One way of obtaining quality assessment results is

```
> qaSummary <- qa(sp)
```

The `qa` object is a list-like structure. As invoked above and currently implemented, `qa` visits all `s_N_export.txt` files in the appropriate directory. It extracts useful information from the files, and summarizes the results into a nested list-like structure.

Evaluating `qa` for a single lane can take several minutes. For this reason a common use case is to evaluate `qa` and save the result to disk for later use, e.g.,

```
> save(qaSummary, file="/path/to/file.rda")
```

A feature of *ShortRead* is the use of *Rmpi* or *multicore* and coarse-grained parallel processing when available. Thus commands such as

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

or

```
> library(multicore)
> qaSummary <- qa(sp)
```

will distribute the task of processing each lane to each of the *Rmpi* workers or *multicore* cores. In the *Rmpi* example, all 8 lanes of a Solexa experiment are processed in the time take to process a single lane. *multicore* may impose significant memory demands, as each core will attempt to load a full lane of data.

The elements of the quality assessment list are suggested by the output:

```
> qaSummary
```

```
class: SolexaExportQA(11)
QA elements (access with qa[["elt"]]):
  readCounts: data.frame(1 3)
  baseCalls: data.frame(1 5)
  readQualityScore: data.frame(1536 4)
  baseQuality: data.frame(94 3)
  alignQuality: data.frame(69 3)
  frequentSequences: data.frame(150 4)
  sequenceDistribution: data.frame(11 4)
  perCycle: list(2)
    baseCall: data.frame(173 4)
    quality: data.frame(648 5)
  perTile: list(2)
    readCounts: data.frame(3 4)
    medianReadQualityScore: data.frame(3 4)
  depthOfCoverage: data.frame(2 4)
  adapterContamination: data.frame(1 1)
```

For instance, the count of reads in each lane is summarized in the `readCounts` element, and can be displayed as

```
> qaSummary[["readCounts"]]
```

```
               read filtered aligned
s_2_export.txt 1000      764     406
```

```
> qaSummary[["baseCalls"]]
```

```
                   A    C    G     T  N
s_2_export.txt 9537 7480 7406 10537 40
```

The `readCounts` element contains a data frame with 1 row and 3 columns (these dimensions are indicated in the parenthetical annotation of `readCounts` in the output of `qaSummary`). The rows represent different lanes. The columns indicated the number of reads, the number of reads surviving the Solexa filtering criteria, and the number of reads aligned to the reference genome for the lane. The `baseCalls` element summarizes base calls in the unfiltered reads.

Other elements of `qaSummary` are more complicated, and their interpretation is not directly obvious. Instead, a common use is to forward the results of `qa` to a report generator.

```
> report(qaSummary, dest="/path/to/report_directory")
```

The report includes R code that can be used to understand how SolexaExportQA-class objects can be processed; reports are generated as HTML suitable for browser viewing.

The functions that produce the report tables and graphics are internal to the package. They can be accessed through calling ShortRead:::functionName where functionName is one of the functions listed below, organized by report section.

Run Summary : .ppnCount, .df2a, .laneLbl, .plotReadQuality

Read Distribution : .plotReadOccurrences, .freqSequences

Cycle Specific : .plotCycleBaseCall, .plotCycleQuality

Tile Performance : .atQuantile, .colorkeyNames, .plotTileLocalCoords, .tileGeometry, .plotTileCounts, .plotTileQualityScore

Alignment : .plotAlignQuality

Multiple Alignment : .plotMultipleAlignmentCount

Depth of Coverage : .plotDepthOfCoverage

Adapter Contamination : .ppnCount

2 Using *ShortRead* for data exploration

2.1 Data I/O

ShortRead provides a variety of methods to read data into R, in addition to `readAligned`.

2.1.1 `readXStringColumns`

`readXStringColumns` reads a column of DNA or other sequence-like data. For instance, the Solexa files `s_N_export.txt` contain lines with the following information:

```
> pattern <- "s_2_export.txt"
> fl <- file.path(analysisPath(sp), pattern)
> strsplit(readLines(fl, n=1), "\t")

[[1]]
 [1] "HWI-EAS88"
 [2] "3"
 [3] "2"
 [4] "1"
```

```
 [5] "451"
 [6] "945"
 [7] ""
 [8] ""
 [9] "CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC"
[10] "YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH"
[11] "NM"
[12] ""
[13] ""
[14] ""
[15] ""
[16] ""
[17] ""
[18] ""
[19] ""
[20] ""
[21] ""
[22] "N"
```

```
> length(readLines(fl))
```

```
[1] 1000
```

Column 9 is the read, and column 10 the ASCII-encoded Solexa Fastq quality score; there are 1000 lines (i.e., 1000 reads) in this sample file.

Suppose the task is to read column 9 as a *DNAStringSet* and column 10 as a *BStringSet*. *DNAStringSet* is a class that contains IUPAC-encoded DNA strings (IUPAC code allows for nucleotide ambiguity); *BStringSet* is a class that contains any character with ASCII code 0 through 255. Both of these classes are defined in the *Biostrings* package. `readXStringColumns` allows us to read in columns of text as these classes.

Important arguments for `readXStringColumns` are the `dirPath` in which to look for files, the `pattern` of files to parse, and the `colClasses` of the columns to be parsed. The `dirPath` and `pattern` arguments are like `list.files`. `colClasses` is like the corresponding argument to `read.table`: it is a *list* specifying the class of each column to be read, or `NULL` if the column is to be ignored. In our case there are 21 columns, and we would like to read in columns 9 and 10. Hence

```
> colClasses <- rep(list(NULL), 21)
> colClasses[9:10] <- c("DNAString", "BString")
> names(colClasses)[9:10] <- c("read", "quality")
```

We use the class of the type of sequence (e.g., *DNAString* or *BString*), rather than the class of the set that we will create (e.g., *DNAStringSet* or *BStringSet*). Applying names to `colClasses` is not required, but makes subsequent manipulation easier. We are now ready to read our file

```
> cols <- readXStringColumns(analysisPath(sp), pattern, colClasses)
> cols

$read
  A DNAStringSet instance of length 1000
      width seq
   [1]    35 CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC
   [2]    35 AGCCTCCCTCTTTCTGAATATACGGCAGAGCTGTT
   [3]    35 ACCAAAAACACCACATACACGAGCAACACACGTAC
   [4]    35 AATCGGAAGAGCTCGTATGCCGGCTTCTGCTTGGA
   [5]    35 AAAGATAAACTCTAGGCCACCTCCTCCTTCTTCTA
   [6]    35 AAAAAAAAAAAGGACACACCATGAGATCACAGGGA
   [7]    35 TAAAAAATTAGCAAAAAACCAAAAATGTAATTGAT
   [8]    35 TAAATCGTGCTGTAACCTTTCCCAACATCTCTGTG
   [9]    35 AATGACCGATAATTAAAAATAAAATCTTTGCATAT
   ...   ... ...
 [992]    35 GAAAAAAAAACAGAACGATGCGTTCATCCACGGCA
 [993]    35 TTATCCCTGGTTTCTCCTTGTGACTCTCTGTTGTC
 [994]    35 AGAGCTTTAGGCAGCTCGGTGTGTCCTTTCTATTC
 [995]    35 TATATTGCCCCCTGCAGCAATGCCCCTTACCCGTC
 [996]    35 GTGGCAGCGGTGAGGCGGCGGGGGGGGGTTGTTTG
 [997]    35 GTCGGAGGTCAGCAAGCTGTAGTCGGTGTAAAGCT
 [998]    35 GTCATAAATTGGACAGTGTGGCTCCAGTATTCTCA
 [999]    35 ATCTACATTAAGGTCAATTACAATGATAAATAAAA
[1000]    35 TTCTCAGCCATTCAGTATTCCTCAGGTGAAAATTC

$quality
  A BStringSet instance of length 1000
      width seq
   [1]    35 YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH
   [2]    35 ZXZUYXZQYYXUZXYZYYZZXXZZIMFHXQSUPPO
   [3]    35 LGDHLILLLLLLLLIGFLLALDIFDILLHFIAECAE
   [4]    35 JJYYIYVSYYYYYYYYSDYYWVUYYNNVSVQQELQ
   [5]    35 LLLILIIIDLLHLLLLLLLLLLLALLLLHLLLLEL
   [6]    35 YYYYYYYWVVMGGUHQHQMUFMICDMCDHQHEDDD
   [7]    35 ZZZZZZZZZZZYZZZZZYZZZZYYZZZZZZZUUUUU
   [8]    35 ZZZZZZZZZUZZUZZZZZZZZZZZZZYZYZZZUUHUH
   [9]    35 ZZZZZZZYZZYZZZZYZZZZZZZZZZZZZXUNUUU
   ...   ... ...
 [992]    35 YYYVVVSSGVSQIGIUSFFYIHLUUHFQXULPLLH
 [993]    35 ZZZZZZZZZZZZZZZZZZZZYZXZZZZZZZSUUJUU
 [994]    35 YIOSMSGSYOSUIYUSUDLIWUQIQQUUUFPLENG
 [995]    35 ZZZZZZZZZZZZZZYZXYZZZXZZZXZZZSZUUUUU
 [996]    35 ZZZZZZZYZZYUYZYUYZKYUDUZIYYODJGUGAA
 [997]    35 ZZZZZZZZZZZZZZZZZZZYZZYXXZYSSXXUUHHQ
 [998]    35 ZZZZZZZZZZZZZZZZYZZZZYZZZZYZZXZUUUUS
```

```
 [999]    35 ZZZZZZZZZZZZYXZYZYZZYZYZZXKZSYXUUNUN
[1000]    35 ZZZZZZZZZZZZZZZYZZZZZZZZZYYSYSZXUUUUU
```

The file has been parsed, and appropriate data objects were created.

A feature of `readXStringColumns` and other input functions in the *ShortRead* package is that all files matching `pattern` in the specified `dirPath` will be read into a single object. This provides a convenient way to, for instance, parse all tiles in a Solexa lane into a single *DNAStringSet* object.

There are several advantages to reading columns as *XStringSet* objects. These are more compact than the corresponding character representation:

```
> object.size(cols$read)

50840 bytes

> object.size(as.character(cols$read))

94280 bytes
```

They are also created much more quickly. And the *DNAStringSet* and related classes are used extensively in *ShortRead*, *Biostrings*, *BSgenome* and other packages relevant to short read technology.

2.1.2 readFastq

`readXStringColumns` should be considered a 'low-level' function providing easy access to columns of data. Another flexible input function is `readFastq`. Fastq files combine reads and their base qualities in four-line records such as the following:

```
> fqpattern <- "s_1_sequence.txt"
> fl <- file.path(analysisPath(sp), fqpattern)
> readLines(fl, 4)

[1] "@HWI-EAS88_1_1_1_1001_499"
[2] "GGACTTTGTAGGATACCCTCGCTTTCCTTCTCCTGT"
[3] "+HWI-EAS88_1_1_1_1001_499"
[4] "]]]]]]]]]]]]Y]Y]]]]]]]]]]]]VCHVMPLAS"
```

The first and third lines are an identifier (encoding the machine, run, lane, tile, x and y coordinates of the cluster that gave rise to the read, in this case). The second line is the read, and the fourth line the per-base quality. Files of this sort can be read in as

```
> fq <- readFastq(sp, fqpattern)
> fq

class: ShortReadQ
length: 256 reads; width: 36 cycles
```

This resulting object (of class ShortReadQ) contains the short reads, their qualities, and the identifiers:

```
> reads <- sread(fq)
> qualities <- quality(fq)
> class(qualities)

[1] "SFastqQuality"
attr(,"package")
[1] "ShortRead"

> id(fq)

  A BStringSet instance of length 256
      width seq
  [1]    24 HWI-EAS88_1_1_1_1001_499
  [2]    23 HWI-EAS88_1_1_1_898_392
  [3]    23 HWI-EAS88_1_1_1_922_465
  [4]    23 HWI-EAS88_1_1_1_895_493
  [5]    23 HWI-EAS88_1_1_1_953_493
  [6]    23 HWI-EAS88_1_1_1_868_763
  [7]    23 HWI-EAS88_1_1_1_819_788
  [8]    23 HWI-EAS88_1_1_1_801_123
  [9]    23 HWI-EAS88_1_1_1_885_419
  ...   ... ...
[248]    23 HWI-EAS88_1_1_1_603_569
[249]    23 HWI-EAS88_1_1_1_718_225
[250]    23 HWI-EAS88_1_1_1_406_412
[251]    23 HWI-EAS88_1_1_1_549_119
[252]    23 HWI-EAS88_1_1_1_693_898
[253]    23 HWI-EAS88_1_1_1_183_559
[254]    23 HWI-EAS88_1_1_1_314_891
[255]    23 HWI-EAS88_1_1_1_884_867
[256]    23 HWI-EAS88_1_1_1_878_444
```

Notice that the class of the qualities is *SFastqQuality*, to indicate that these are quality scores derived using the Solexa convention, rather than ordinary *BStringSet* objects.

The object has essential operations for convenient manipulation, for instance simultaneously forming the subset of all three components:

```
> fq[1:5]

class: ShortReadQ
length: 5 reads; width: 36 cycles
```

2.1.3 Additional input functions

ShortRead includes additional functions to facilitate input. For instance, `readPrb` reads Solexa `_prb.txt` files. These files contain base-specific quality information, and `readPrb` returns an *SFastqQuality*-class object representing the fastq-encoded base-specific quality scores of all reads.

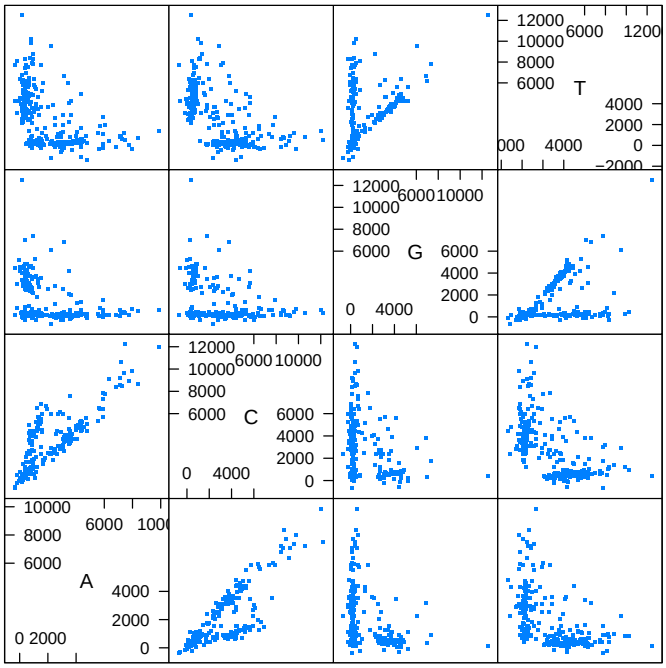
As a second example, the `s_N_LLLL_int.txt` files in the `imageAnalysisPath` directory contain lines, one line per read, of nucleotide intensities. Each line contain lane, tile, X and Y coordinate information, followed by quadruplets of intensity values. There are as many quadruplets as there are cycles. Each quadruplet represents the intensity of the `A`, `C`, `G`, and `T` nucleotide at the corresponding cycle. These (and their error estimates, if available), are input with

```
> int <- readIntensities(sp, withVariability=FALSE)
> int

class: SolexaIntensity
dim: 256 4 36
readInfo: SolexaIntensityInfo
intensity: ArrayIntensity
measurementError: not available
```

An interesting exercise is to display the intensities at cycle 2 (below) and to compare these to cycle, e.g., 30.

```
> print(splom(intensity(int)[[,,2]], pch=".", cex=3))
```



Scatter Plot Matrix

Additional files can be parsed using standard R input methods.

2.2 Sorting

Short reads can be sorted using `srsort`, or the permutation required to bring the short read into lexicographic order can be determined using `srorder`. These functions are different from `sort` and `order` because the result is independent of the locale, and they operate quickly on *DNAStringSet* and *BStringSet* objects.

The function `srduplicated` identifies duplicate reads. This function returns a logical vector, similar to `duplicated`. The negation of the result from `srduplicated` is useful to create a collection of unique reads. An experimental scenario where this might be useful is when the sample preparation involved PCR. In this case, replicate reads may be due to artifacts of sample preparation, rather than differential representation of sequence in the sample prior to PCR.

2.3 Summarizing read occurrence

The `tables` function summarizes read occurrences, for instance,

```
> tbls <- tables(aln)
> names(tbls)

[1] "top"          "distribution"
```

```
> tbls$top[1:5]

GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTAGA
                                  10
GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAA
                                   5
GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTTGA
                                   3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                                   2
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
                                   2

> head(tbls$distribution)

  nOccurrences nReads
1            1    972
2            2      5
3            3      1
4            5      1
5           10      1
```

The `top` component returned by `tables` is a list tallying the most commonly occurring sequences in the short reads. Knowledgeable readers will recognize the top-occurring read as a close match to one of the manufacturer adapters.

The `distribution` component returned by `tables` is a data frame that summarizes how many reads (e.g., 972) are represented exactly 1 times.

2.4 Finding near matches to short sequences

Facilities exist for finding reads that are near matches to specific sequences, e.g., manufacturer adapter or primer sequences. `srdistance` reports the edit distance between each read and a reference sequence. `srdistance` is implemented to work efficiently for reference sequences whose length is of the same order as the reads themselves (10's to 100's of bases). To find reads close to the most common read in the example above, one might say

```
> dist <- srdistance(sread(aln), names(tbls$top)[1])[[1]]
> table(dist)[1:10]

dist
 0  1  2  3  4  7  9 10 12 13
10  7 11  2  1  1  1  1  3  2
```

'Near' matches can be filtered from the alignment, e.g.,

```
> alnSubset <- aln[dist>4]
```

A different strategy can be used to tally or eliminate reads that consist predominantly of a single nucleotide. `alphabetFrequency` calculates the frequency of each nucleotide (in DNA strings) or letter (for other string sets) in each read. Thus one could identify and eliminate reads with more than 30 adenine nucleotides with

```
> countA <- alphabetFrequency(sread(aln))[,"A"]
> alnNoPolyA <- aln[countA < 30]
```

`alphabetFrequency`, which simply counts nucleotides, is much faster than `srdistance`, which performs full pairwise alignment of each read to the subject.

Users wanting to use R for whole-genome alignments or more flexible pairwise aligment are encouraged to investigate the *Biostrings* package, especially the *PDict* class and `matchPDict` and `pairwiseAlignment` functions.

2.5 The `coverage` function

The `coverage` function provides a way to summarize where reads align on a reference sequence. The idea is that the aligned reads, or under some analyses the extension of those aligned reads by an amount meant to estimate the actual fragment size, 'pile up' on top of nucleotide positions in the reference sequence. A convenient summary of the alignment of many reads is thus a vector describing the depth of the pile at each position in the reference sequence. A typical work flow invokes `coverage` on an instance of the *AlignedRead* class obtained from `readAligned`; additional methods offering greater control operate on *IRanges* directly. The `coverage` methods returns a run-length encoding of the pile-up (or a list of such run length encodings). The run-length encoding returned by `coverage` is a space-efficient representation; the long integer vector can be recovered with `as.integer`.

There are complicated issues associated with use of `coverage`, relating to how software reports the 'position' of an alignment, especially on the minus strand. These issues are illustrated in figure 1. In the figure, the two strands are represented by `....|`, aligned reads by `+++`, and extensions by `---`. The idea is that 5-nucleotide reads have been aligned to a reference sequence, and the alignment extended by 10 nucleotides. In the 'leftmost' notation (used by ELAND) and assuming that the reference sequence is always numbered in relation to the plus strand and indexed starting at 1 (`readAligned` translates reported alignment positions so they are indexed from 1), the reported position is 15 for the alignments on either the plus or the minus strand. In contrast the 'fiveprime' scheme the alignment to the plus strand is 15, and to the minus strand 19. This is the scheme used by MAQ, for instance.

The default behavior of `coverage` is to use the 'leftmost' coordinate system. This is appropriate for data derived from ELAND.

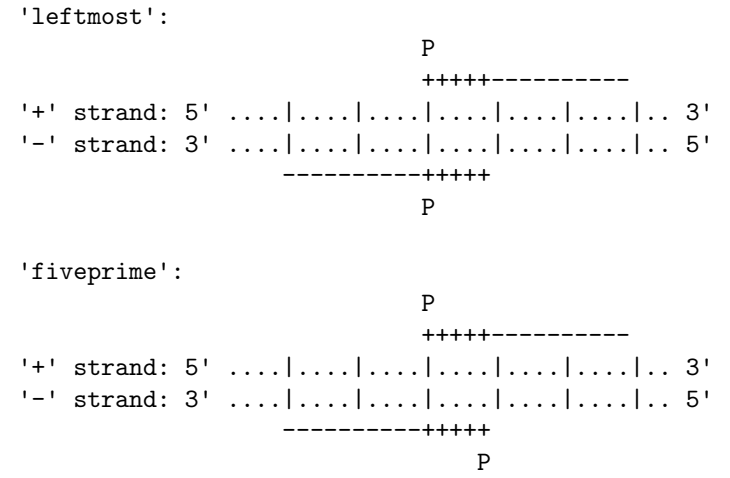


Figure 1: Alignment schemes used by `coverage`. `+++` represents the read and `--` the extension. `P` is the alignment position as recorded under the corresponding leftmost or fiveprime schemes.

3 Advanced features

3.1 The `pattern` argument to input functions

Most *ShortRead* input functions are designed to accept a directory path argument, and a `pattern` argument. The latter is a grep-like pattern (as used by, e.g., `list.files`). Many input functions are implemented so that all files matching the pattern are read into a single large input object. Thus the `s_N_LLLL_seq.txt` files consist of four numeric columns and a fifth column corresponding to the short read. The following code illustrates the file structure and inputs the final column into a *DNAStringSet*:

```
> seqFls <- list.files(baseCallPath(sp), "_seq.txt", full=TRUE)
> strsplit(readLines(seqFls[[1]], 1), "\t")

[[1]]
[1] "1"
[2] "1"
[3] "109"
[4] "548"
[5] "TTGTTTTCATGTGATTTTAAAAATGTATTTGTTTGT"

> colClasses <- c(rep(list(NULL), 4), "DNAString")
> reads <- readXStringColumns(baseCallPath(sp), "s_1_0001_seq.txt",
+                             colClasses=colClasses)
```

The more general pattern

```
> reads <- readXStringColumns(baseCallPath(sp), "s_1_.*_seq.txt",
+                             colClasses=colClasses)
```

inputs all lane 1 tile files into a single *DNAStringSet* object.

3.2 srapply

Solexa and other short read technologies often include many files, e.g., one `s_L_NNNN_int.txt` file per tile, 300 tiles per lane, 8 lanes per flow cell for 2400 `s_L_NNNN_int.txt` files per flow cell. A natural way to extract information from these is to write short functions, e.g., to find the average intensity per base at cycle 12.

```
> calcInt <- function(file, cycle, verbose=FALSE)
+ {
+     if (verbose)
+         cat("calcInt", file, cycle, "\n")
+     int <- readIntensities(dirname(file), basename(file),
+                            intExtension="", withVariability=FALSE)
+     apply(intensity(int)[,,12], 2, mean)
+ }
```

One way to apply this function to all intensity files in a Solexa run is

```
> intFls <- list.files(imageAnalysisPath(sp), ".*_int.txt$", full=TRUE)
> lres <- lapply(intFls, calcInt, cycle=12)
```

The files are generally large and numerous, so even simple calculations consume significant computational resources. The `srapply` function is meant to provide a transparent way to perform calculations like this distributed over multiple nodes of an MPI cluster, or across multiple cores of a single machine. Thus

```
> srres <- srapply(intFls, calcInt, cycle=12)
> identical(lres, srres)
```

```
[1] TRUE
```

evaluates the function as `lapply`, whereas

```
> library("Rmpi")
> mpi.spawn.Rslaves(nsl=16)
> srres <- srapply(intFls, calcInt, cycle=12)
> mpi.close.Rslaves()
```

distributes the calculation over available workers, while

```
> library(multicore)
> srres <- srapply(intFls, calcInt, cycle=12)
```

distributes tasks across cores of a single machine. The result is a speedup approximately inversely proportional to the number of available compute nodes or cores; memory requirements for the *multicore* approach may be substantial.

```
> toLatex(sessionInfo())
```

- R version 2.14.1 (2011-12-22), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Biostrings 2.22.0, GenomicRanges 1.6.4, IRanges 1.12.5, RColorBrewer 1.0-5, Rsamtools 1.6.3, ShortRead 1.12.4, lattice 0.20-0, latticeExtra 0.6-19
- Loaded via a namespace (and not attached): BSgenome 1.22.0, Biobase 2.14.0, RCurl 1.8-0, XML 3.6-2, bitops 1.0-4.1, grid 2.14.1, hwriter 1.3, rtracklayer 1.14.4, tools 2.14.1, zlibbioc 1.0.0

Table 1: The output of `sessionInfo` on the build system after running this vignette.

4 Conclusions and directions for development

ShortRead provides tools for reading, manipulation, and quality assessment of short read data. Current facilities in *ShortRead* emphasize processing of single-end Solexa data.

Development priorities in the near term include expanded facilities for importing key file types from additional manufactures, more extensive quality assessment methodologies, and development of infrastructure for paired-end reads.