An Introduction to *ShortRead*

Martin Morgan

25 July, 2008

```
> 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, Helicos, SOLiD, and related technologies. This vignette introduces key functionality.

The package is still very much in development. 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/Rinst550125137/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/Rinst550125137/ShortRead/extdata/Data/C1-36Firecrest"

> analysisPath(sp)

[1] "/tmp/Rinst550125137/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. *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 ... y filtering
```

`readAligned` 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.

Currently supported alignment files include:

SolexaExport the Solexa 'export' file format described in the Solexa pipeline version 0.3 documentation.

MAQMapview the MAQ 'mapview' format described at `http://maq.sourceforge.net/maq-manpage.shtml#5`, as viewed 3 May, 2008.

MAQMap the MAQ binary 'map' format described at `http://maq.sourceforge.net/maq-manpage.shtml#5`, as viewed 3 May, 2008.

Other parser contributions are welcome. Paired end read support is not yet available.

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 ZZZZZZZZZZZZZZZZZZZYZZYXXZYSSXXUUHHQ
 [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 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* package.

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; the level `""` corresponds to unaligned reads) and tabulated using familiar R functions.

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

```
[1] "factor"
```

```
> levels(whichStrand)
```

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

```
> table(whichStrand)
```

```
whichStrand
  -   +   *
203 203   0
```

This shows that about NA percent of reads were not aligned (level `""`).

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 and varMetadata description:
    run: Analysis pipeline run
    lane: Flow cell lane
    ...: ...
    filtering: Read successfully passed filtering?
    (6 total)
```

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?
```

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 ... y filtering
```

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* and coarse-grained parallel processing when available. Thus commands such as

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

will distribute the task of processing each lane to each of the *Rmpi* workers. With *Rmpi*, all 8 lanes of a Solexa experiment are processed in the time take to process a single lane.

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

```
> qaSummary
```

```
class: SolexaExportQA(9)
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)
```

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/qa_report.pdf")
```

The report includes R code that can be used to understand how SolexaExportQA-class objects can be processed.

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 underlying object, *DNAString* or *BString*, rather than the class of the object we will create, e.g., *DNAStringSet*. Applying names to `colClasses` is not required but makes subsequent manipulation easy. 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 ZZZZZZZZZUZZUZZZZZZZZZZZZYZYZZZUUHUH
   [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 ZZZZZZZZZZZZZZZYZZZZZZZZYYSYSZXUUUUU
```

The file is parsed and appropriate data objects 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)

[1] 46592

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

[1] 94280
```

They are read in 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.

Additional files can be parsed using standard R input methods. For instance, 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. Thus

```
> intFile <- list.files(imageAnalysisPath(sp), "s_1_0001_int.txt",
+     full = TRUE)
> strsplit(readLines(intFile, 1), "\t")[[1]][1:6]
```

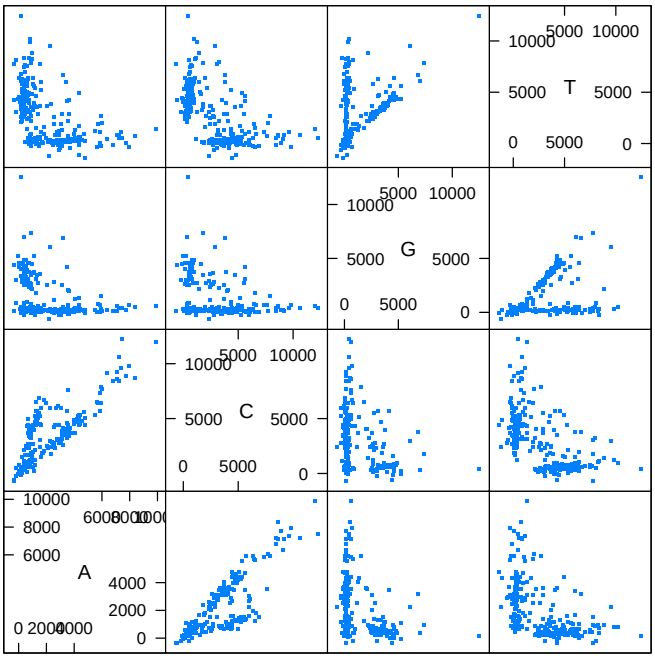
```
[1] "1"
[2] "1"
[3] "109"
[4] "548"
[5] " 409.0  504.5  475.0 11120.8"
[6] " 880.8 3231.2  464.8 7933.4"
```

```
> intDf <- read.table(intFile)
> dim(intDf)
```

```
[1] 256 148
```

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

```
> c2 <- intDf[, 2 * 4 + 1:4]
> colnames(c2) <- c("A", "C", "G", "T")
> print(splom(c2, pch = ".", cex = 3))
```



Scatter Plot Matrix

2.2 Sorting

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

The function `srduplicated` identifies duplicate reads. This function returns a logical vector much like `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 sample preparation involves 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 (short!) reference sequence. To find reads close to the most common read in the example above, one might

```
> 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 are 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 A 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 alignemnt 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 `pileup`

`pileup` provides a convenient way to summarize where reads align on reference sequences. `pileup` uses alignments obtained from other sources, e.g., `readAligned` or the `PDict` family of functions in *Biostrings*.

3 Advance 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 as 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., 1 `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(filename, cycle, verbose = FALSE) {
+     if (verbose)
+         cat("calcInt", filename, cycle, "\n")
+     c12 <- read.table(filename)[, cycle * 4 +
+         1:4]
+     colnames(c12) <- c("A", "C", "G", "T")
+     colMeans(c12)
+ }
```

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. 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, resulting in a speedup approximately inversely proportional to the number of available compute nodes.

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.