Analysis of high-throughput short reads using R and BioConductor

Patrick Aboyuon & Martin Morgan
Fred Hutchinson Cancer Research Center
Seattle, WA 98008

4 September, 2008

Contents

1 Introduction

Short read technologies

- 100's of thousands to 10's of millions of short (100's to 10's of nucleotides) DNA sequences.

- Examples: Solexa (Illumina), 454 (Roche), SOLiD (Applied Biosystems), Helicos.
- More detail: Solexa
  - Biological preparation: e.g., enrichment, ChIP, ...
  - Sample processing: fragment ('random'); attach adapters, PCR primers; add to flow cell; amplify; sequence

Vendor and third-party analysis:

- Image interpretation; base calls; short read alignment / assembly / SNP discovery / ...
- Examples: ELAND (Solexa alignment); MAQ

R and BioConductor

- R: interactive, extensible statistical programming language.
- BioConductor: collection of R packages for analysis of high-throughput (microarray, flow cytometry, short read, and other) technologies.
- Especially suited for research, development, and popularization of new methods.

Use cases for short reads in R and BioConductor

- Input, exploration, and data management of aligned or raw sequences.
- Alignment, especially to understand statistical aspects of the data or to address unique research questions.
- Advanced tool development.
- Caveat: BioConductor tools do not currently implement one-click work flows.

1.1 Exercises

Exercise 1
Copy the OS-specific folder hierarchy from the memory stick to a convenient location on your computer. The following assumes that you copied the hierarchy to a folder named `MGED-2008` (it avoids confusion in R to use / for the file path separator on Windows).

Exercise 2
Install R.

1. *Windows and Mac: double click on the installer and follow directions.*
2. *Linux: consult with tutorial assistants.*

Exercise 3
Check installation / a first R session.

1. *Start R (e.g., double-click on the appropriate icon, or select from the 'start' menu.*

2. *Enter the text that appears after the* `>` *and confirm that you are using R version* `2.8.0 Under development (unstable)` *:*

```
> sessionInfo()

R version 2.8.0 Under development (unstable) (2008-08-22 r46416)
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

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

3. *Find help about a function, e.g.,* `plot` *with the command*

```
> ?plot
```

4. *Load a special purpose library (in this case, for more advanced plotting)*

```
> library(lattice)
```

End the R session (select `n` *when prompted to save the session).*

```
> q()
```

Exercise 4
Install essential BioConductor packages (all OS).

1. *Start a new R session (e.g., by double clicking on the appropriate icon or selecting from the 'start' menu)*

2. *Evaluate the following command, replacing your operating system and file path as appropriate:*

```
> install.packages(c("ShortRead", "BSgenome.Mmusculus.UCSC.mm9"),
+                  repos="file:///home/mtmorgan/MGED-2008/Linux/repos")
```

On Windows, specify `repos` *as, for instance* `file:///c:/MGED-2008/Windows/repos`*.*

3. *Note: the usual way to install packages is from the internet, following instructions at* `http://bioconductor.org/install`*.*

Verify the installation by trying to load one of the installed packages, e.g.,

```
> library(ShortRead)
```

Exercise 5
Further activities.

1. *BioConductor packages have* vignettes *that provide an integrated narrative of how the package is used. Use* `openVignette()` *to explore the* `Overview` *vignette in the ShortRead package. If the vignette is not available on your computer, see the next instruction.*

2. *The BioConductor web site summarizes available packages. Visit* `http://bioconductor.org/packages/devel/Software.html` *to discover software available for the development version of R. Can you find the ShortRead package and its vignette on the web?*

2 I/O and quality assessment of aligned reads

In this section we read in and assess the quality of sequences produced and aligned by ELAND. We'll address the following general questions:

- What information can we easily import and access in R?
- Can we assess the quality of the experiment?

2.1 Input

Load the ShortRead package.

```
> library(ShortRead)
```

Create a reference to the data on disk

```
> sp <- SolexaPath("MGED-2008/extdata/Solexa")
> sp
```

```
class: SolexaPath
experimentPath: MGED-2008/extdata/Solexa
dataPath: Data
scanPath: NA
imageAnalysisPath: C1-35Firecrest
baseCallPath: Bustard
analysisPath: GERALD
```

Read the aligned data in to R

```
> aln <- readAligned(sp, pattern="s_1_export.txt")
```

Other input methods are available, for other Solexa files and for non-Solexa technologies, e.g., `readFastq`, `readXStringColumns`.

2.2 Preliminary data exploration

Here is a summary of the data we have read in:

```
> aln
```

```
class: AlignedRead
length: 100000 reads; width: 35 cycles
chromosome: chrUn_random.fa chr1.fa ... 255:255:255 NM
position: 5251632 163068613 ... NA NA
strand: F R ...
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

Reads are accessed with `sread`.

```
> sread(aln)
```

```
  A DNAStringSet instance of length 100000
         width seq
     [1]    35 TTTCAGTTTTCTCGCCTTATTCCATGTCCTACAGT
     [2]    35 TAGACTGCTGCCTAGCAAGCCTTAAGGATTCTTCT
     [3]    35 TGCAAGAAGTGGAATACAAAACAAAGGCTTAGAAT
     [4]    35 AAAATGAGAAACATCCACTTGACTCCTTGAAAAAT
     [5]    35 TGGGCTGACGTCATGCCTGAGCTGTCACGAGCAGA
     [6]    35 GCGAGGAAAACTGAAAAAGGTGGAAAATTTAGAAA
     [7]    35 GATGAACAAGAGTTTACCAAAAGGTCAAAATGAAA
     [8]    35 GAAAAATGAGAAATGCACACTGTAGGACCTGGAAT
     [9]    35 GTTTTAGGAACTCTCCTTATAGAAGAAACAACTCG
     ...   ... ...
 [99992]    35 AAAATCACTCACAACCGTGAACAATGAGACATGCC
 [99993]    35 ACATACCCCCCCAACTCCACCCTACCTAACACCTC
 [99994]    35 TAACTCAGCCAGCCACAAAGAGAAACAACCAAACC
 [99995]    35 CATGTCGCCCCCTTCTAATGGTCCTCAGCCAGGTG
 [99996]    35 CTGAATACCACCTGTCTCTAGCTCACAGCACACTG
 [99997]    35 CCTCTCTCCAAAATACCATCACACTCCCCCCCCAC
 [99998]    35 AAAAAAAAAACTTCCTCTTCCCCATTTCTTTCTTT
 [99999]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[100000]    35 AAATCAACATAACACATATATCCTCCACCACATCA
```

'Quality' scores (base call confidence, defined in vendor-specific ways) are accessed with `quality`

```
> quality(aln)
```

`quality` measures for Solexa are typically reported as modified 'fastq' scores, where each base is assigned an ASCII symbol representing its quality. Higher-valued ASCII characters (e.g., later in the alphabet) correspond to bases with higher quality.

```
> sqchars <- SolexaQuality("ABCDEFGHIJKLMNOPQRSTUVWXYZ[")
> sqchars

  A SolexaQuality instance of length 1
    width seq
[1]    27 ABCDEFGHIJKLMNOPQRSTUVWXYZ[

> ## associated numeric equivalents
> as.integer(sqchars)

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27

> ## associated miscall error probabilities
> round(as.numeric(sqchars), 4)

 [1] 0.4427 0.3869 0.3339 0.2847 0.2403 0.2008 0.1663 0.1368
 [9] 0.1118 0.0909 0.0736 0.0594 0.0477 0.0383 0.0307 0.0245
[17] 0.0196 0.0156 0.0124 0.0099 0.0079 0.0063 0.0050 0.0040
[25] 0.0032 0.0025 0.0020
```

We can summarize essential alignment information by accessing data and applying standard R functions. For instance, alignments to forward and reverse strands are tabulated with

```
> table(strand(aln))

          F     R
50705 24661 24634
```

about 1/2 the reads were not aligned (the first column of numbers); a result often seen is that more reads align to the forward than to the reverse strand.

Vendor-specific information on alignments is stored in a data frame-like structure that can be displayed with

```
> alignData(aln)

An object of class "AlignedDataFrame"
  readName: 1, 2, ..., 100000  (100000 total)
  varLabels and varMetadata description:
    run: Analysis pipeline run
    lane: Flow cell lane
    ...: ...
    filtering: Read successfully passed filtering?
    (6 total)

> head(pData(alignData(aln)))
```

```
  run lane tile   x   y filtering
1   1    1    1  98 349         Y
2   1    1    1  83 277         Y
3   1    1    1  72 446         N
4   1    1    1 123 436         Y
5   1    1    1 114 969         Y
6   1    1    1 343 187         Y
```

`pData(alignData(aln))` extracts the vendor-specific alignment data. `head` is an R function to display the first few lines of the data. ELAND provides information on the software run number, the lane, tile, x, and y coordinates of each read, and whether the read survivived ELAND-specific filtering criteria (the ELAND filtering criteria are described in the ELAND manual; default filtering is based on aspects of quality of the initial 12 bses).

Objects can be subset to contain only specific reads. For instance,

```
> filterOk <- alignData(aln)[["filtering"]] == "Y"
```

creates a logical vector with value `TRUE` when the read passed ELAND filtering, and `FALSE` otherwise. Create a new set of aligned reads containing just those reads passing filtering with

```
> aln1 <- aln[filterOk]
> aln1
```

```
class: AlignedRead
length: 59975 reads; width: 35 cycles
chromosome: chrUn_random.fa chr1.fa ... NM 255:255:255
position: 5251632 163068613 ... NA NA
strand: F R ...
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

Many other possibilities for subsetting data are possible.

2.3 Quality assessment

ShortRead provides quality assessment for Solexa (currently) and other (planned) technologies. The assessment is meant to supplement, rather than replicate, quality assessment measures provided by the vendor or other software. The following commands perform quality assessment and generate a pdf-style report; the commands require too much data for this tutorial, so cannot be executed directly.

```
> qa <- qa(sp)                          # perform QA
> rpt <- report(qa)                     # format a report
```

The result of these commands is a PDF document summarizing the runs (HTML-based reports are planned). A sample is included at `MGED-2008/doc/qa_-080623_080728.pdf`; the `qa` object can be read in with the command

```
> load("MGED-2008/data/qa_080623_080706.rda")
```

Some features of the report, which is more-or-less typical of the runs seen in our new sequencing facility, include:

- Lanes produced between 5.5 and 7.7 million reads; less than 1/2 of the reads aligned to the reference genome.
- About 5% of nucleotides were called `N`, i.e., uncertain; `A` and `C` nucleotides were generally called more frequently than `G` and `T`. The importance of this needs to be assessed in relation to the reference genome (mouse, in this case) and resequencing target.
- Some reads were present at very high frequency. These correspond to adapter sequences (likely an artifact of sample preparation) or other obviously anomalous reads (e.g., all-`A`). Conversely, some reads were encountered only one or a couple of times, these likely represent sequencing errors.
- Nucleotide frequencies were not constant across cycles, e.g., the `C` nucleotide was more common at the end than at the beginning of the read. This likely represents a technological limitation, perhaps related to low signal intensity.
- Read count and quality exhibit spatial effects in each lane; per tile read quality is inversely related to read count. A few tiles failed completely.

2.4 Exercises

Perform the following, using the commands sketched above as a guide. Solutions appear in blue, below.

Exercise 6
Use `SolexaPath` to assign a value to an R object `sp` that is a reference to the portion of the Solexa file hierarchy distributed with the tutorial (use `MGED-2008/extdata/Solexa` as the `experimentPath`).

Use the help page `?SolexaPath` and the hints suggested by the display of `sp` to determine the `analysisPath`, i.e., the location in the file hierarchy where the results of ELAND analysis are located.

Use `baseCallPath` and `list.files` (consulting its help page, if necessary) to list all files in the base call directory. List only files matching the pattern `_seq.txt`; these files contain the lane, tile, x, and y coordinates, and base sequence of each read.

```
> sp <- SolexaPath("MGED-2008/extdata/Solexa")
> analysisPath(sp)

[1] "MGED-2008/extdata/Solexa/Data/C1-35Firecrest/Bustard/GERALD"
```

```
> list.files(baseCallPath(sp), "_seq.txt")

[1] "s_1_0001_seq.txt" "s_5_0001_seq.txt"
```

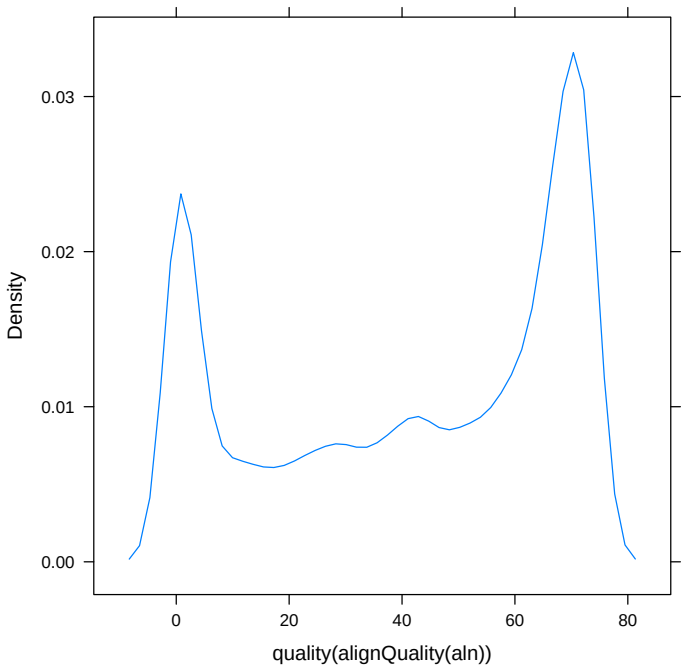
Exercise 7

Use `readAligned`*, including its* `pattern` *argument, to input the reads of lane 1. Extract the reads from the result of* `readAligned` *using* `sread`*.*

Aligned reads are assigned a numeric quality score. The quality score depends on the algorithm used, and in this case the Solexa ELAND documentation should be consulted for precise definition. For our purposes, we'll say that ELAND reports the 'best' alignment with up to two mismatches, of each read against a reference genome. The quality of the alignment is reported as a numeric value, with high scores corresponding to better alignment. Use `alignQuality` *and* `quality` *to extract the quality of each alignment. Plot the read quality with* `hist`*, and with* `densityplot` *(hint:* `densityplot` *is in the lattice pacakge; consult the help pages for insight into how to invoke these functions). Should all alignments be considered equally good?*

```
> aln <- readAligned(sp, "s_1_export.txt$")
> sread(aln)

  A DNAStringSet instance of length 100000
         width seq
     [1]    35 TTTCAGTTTTCTCGCCTTATTCCATGTCCTACAGT
     [2]    35 TAGACTGCTGCCTAGCAAGCCTTAAGGATTCTTCT
     [3]    35 TGCAAGAAGTGGAATACAAAACAAAGGCTTAGAAT
     [4]    35 AAAATGAGAAACATCCACTTGACTCCTTGAAAAAT
     [5]    35 TGGGCTGACGTCATGCCTGAGCTGTCACGAGCAGA
     [6]    35 GCGAGGAAAACTGAAAAAGGTGGAAAATTTAGAAA
     [7]    35 GATGAACAAGAGTTTACCAAAAGGTCAAAATGAAA
     [8]    35 GAAAAATGAGAAATGCACACTGTAGGACCTGGAAT
     [9]    35 GTTTTAGGAACTCTCCTTATAGAAGAAACAACTCG
     ...   ... ...
 [99992]    35 AAAATCACTCACAACCGTGAACAATGAGACATGCC
 [99993]    35 ACATACCCCCCCAACTCCACCCTACCTAACACCTC
 [99994]    35 TAACTCAGCCAGCCACAAAGAGAAACAACCAAACC
 [99995]    35 CATGTCGCCCCCTTCTAATGGTCCTCAGCCAGGTG
 [99996]    35 CTGAATACCACCTGTCTCTAGCTCACAGCACACTG
 [99997]    35 CCTCTCTCCAAAATACCATCACACTCCCCCCCCAC
 [99998]    35 AAAAAAAAAACTTCCTCTTCCCCATTTCTTTCTTT
 [99999]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[100000]    35 AAATCAACATAACACATATATCCTCCACCACATCA

> qscores <- quality(alignQuality(aln))
> head(qscores)
```

```
[1]  3 43 37  2 73 NA

> ## hist(qscores)
> library(lattice)
> densityplot(qscores, plot.points=FALSE)
```



Exercise 8
Review the qa report, `MGED-2008/doc/qa_080623_080728.pdf`. Can you find support for each of the statements in section 2.3? Are there features of the data that are surprising, or worth further investigation?

3 Examining short reads

This section will delve a little more deeply into working with short reads. We'll explore:

- Summarizing nucleotide frequencies
- Removing anomalous reads prior to subsequent analysis.

3.1 Summarizing nucleotide frequencies

ShortRead and Biostrings have a number of useful functions for summarizing sequences. `alphabetFrequency` summarizes nucleotide use, either over all reads or on a per-read basis

```
> alphabetFrequency(sread(aln), collapse=TRUE)   # over all reads
```

```
     A      C      G      T      M      R      W      S
882973 916323 774732 865105      0      0      0      0
     Y      K      V      H      D      B      N      -
     0      0      0      0      0      0  60867      0
     +
     0
```

```
> alf <- alphabetFrequency(sread(aln))           # each read
> head(alf)
```

```
      A  C  G  T M R W S Y K V H D B N - +
[1,]  5 10  4 16 0 0 0 0 0 0 0 0 0 0 0 0 0
[2,]  8  9  7 11 0 0 0 0 0 0 0 0 0 0 0 0 0
[3,] 17  4  8  6 0 0 0 0 0 0 0 0 0 0 0 0 0
[4,] 16  7  4  8 0 0 0 0 0 0 0 0 0 0 0 0 0
[5,]  7  9 12  7 0 0 0 0 0 0 0 0 0 0 0 0 0
[6,] 18  2 10  5 0 0 0 0 0 0 0 0 0 0 0 0 0
```

`alphabetByCycle` summarizes alphabet use (i.e., nucleotides) by cycle

```
> abc <- alphabetByCycle(sread(aln))
> abc[,1:2]                                  # cycles 1:2
```

```
        cycle
alphabet  [,1]  [,2]
       A 28604 32010
       C 23404 20881
       G 27126 21717
       T 20593 23167
       M     0     0
       R     0     0
       W     0     0
       S     0     0
       Y     0     0
       K     0     0
       V     0     0
       H     0     0
       D     0     0
       B     0     0
       N   273  2225
       -     0     0
       +     0     0
```

```
> abc[,34:35]                              # cycles 34:35
```

```
        cycle
alphabet  [,1]  [,2]
       A 24200 26353
       C 26595 26286
       G 21539 19975
       T 26416 26261
       M     0     0
       R     0     0
       W     0     0
       S     0     0
       Y     0     0
       K     0     0
       V     0     0
       H     0     0
       D     0     0
       B     0     0
       N  1250  1125
       -     0     0
       +     0     0
```

The `tables` function provides two summaries of common reads.

```
> tbls <- tables(aln)
> ## most prevalent
> head(tbls[["top"]], 3)
```

```
GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAA
                                  649
GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTAGA
                                  472
ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
                                  381
```

```
> ## times a read occurs
> head(tbls[["distribution"]], 4) # e.g., 95721 singleton reads
```

```
  nOccurrences nReads
1            1  95721
2            2    337
3            3     85
4            4     39
```

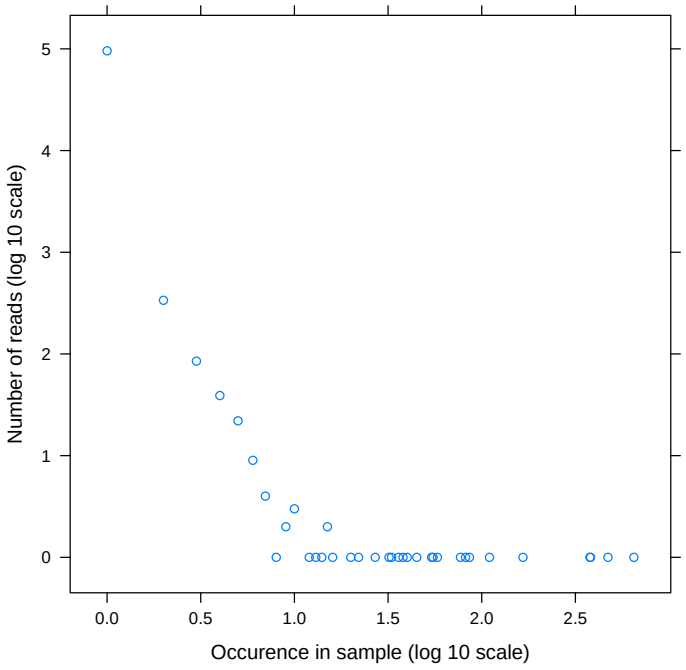
```
> tail(tbls[["distribution"]], 4) # e.g., 1 read represented 649 times
```

```
   nOccurrences nReads
33          378      1
```

```
34          381       1
35          472       1
36          649       1
```

One way of visualizing the distribution of reads is in a plot of the number of reads that are rpresented once, twice, and so on (I call these 'hoover' plots, after the English term for what North Americans refer to as a vacuum cleaner):

```
> print(xyplot(log10(nReads) ~ log10(nOccurrences),
+        tbls[["distribution"]],
+        ylab="Number of reads (log 10 scale)",
+        xlab="Occurence in sample (log 10 scale)"))
```



These plots are quite informative. If reads represent a highly replicated uniform random sample of a region of DNA, then read counts would follow a binomial distribution centered around the average coverage. Instead, the distribution in the 'head' of the hoover is much broader than binomial expectation, suggesting that additional sequence features (e.g., local GC content) influence how often a particular read is represented. The linear 'handle' (sometimes distinctly concave) suggests a scale-free process to describe sequencing error.

3.2 'Cleaning' reads

Reads will often need to be 'cleaned' in a variety of ways, especially to remove experiment-specific artifacts produced during sample preparation or amplification. We can easily remove reads with uncalled bases (which can cause subtle ambiguity in alignment algorithms, for instance)

```
> length(aln)
```

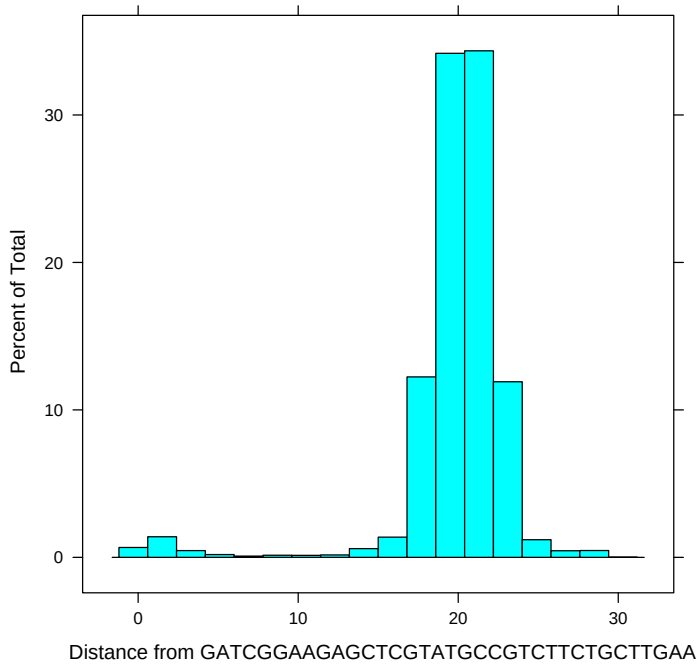
```
[1] 100000
```

```
> clean <- aln[alf[,"N"] == 0]
> length(clean)
```

```
[1] 96818
```

In our sample, a particular sequence (likely a Solexa adapter) is represented by many copies. We can find reads 'near' (in terms of edit distance) to this sequence, and remove suspect reads.

```
> subj <- "GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAA"
> dist <- srdistance(sread(clean), subj)[[1]]
> histogram(dist, xlab=paste("Distance from", subj))
```



We can use facilities in R to discover features of these reads. For instance, were the reads that were 'close' to the adapater filtered by ELAND?

```
> table(dist < 5, alignData(clean)[["filtering"]])

            N     Y
  FALSE 36479 57892
  TRUE    386  2061
```

Finally, we can ask whether we can spot any remaining problems (it looks like there are other problems!)

```
> cleaner <- clean[dist > 5]
> head(tables(cleaner)[["top"]], 3)

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                                 378
GATCGGAAGAGCGGTTCAGCAGGAATGCCGAGATC
                                  32
GTTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAG
                                  13
```

A different strategy, especially if uncertainty about biased PCR is a concern, is to select only a single copy of each read

```
> uclean <- clean[!srduplicated(sread(clean))]
> length(uclean)

[1] 93861
```

Functions related to sorting include `srsort`, `srrank`, and `srorder`.

3.3 Exercises

This set of exercises will use some of the tools suggested above to look at how nucleotied frequency and base quality change across reads.

Exercise 9
Use `SolexaPath` to make an R object `sp` referencing the demo Solexa file hierarch. Use this to read in the sample of aligned data from lane 1.

```
> ## use SolexaPath and readAligned
> sp <- SolexaPath("MGED-2008/extdata/Solexa")
> aln <- readAligned(sp, "s_1_export.txt")
```

Exercise 10
Summarize alphabet frequency over all reads, and per cycle. Assign the per cycle value to an R object `abc`. Use the R functions `head`, `class`, `length`, and `dim` (for `abc`) to explore the object you have created.

```
> alf <- alphabetFrequency(sread(aln), collapse=TRUE)
> class(alf)

[1] "integer"

> length(alf)

[1] 17

> abc <- alphabetByCycle(sread(aln))
> class(abc)

[1] "matrix"

> dim(abc)

[1] 17 35
```

Exercise 11

Extract the quality scores from the aligned reads, using the R function `quality`*. Convert the quality scores into a matrix of numeric values, and keep only cycles 1 through 30. Do this using*

```
> Q <- as(quality(aln), "matrix")[,1:30]
```

Explore the properties of `Q` *with* `class`*,* `dim`*, and* `head`*. Can you determine what the dimensions of* `Q` *correspond to?*

According to Solexa documentation, average quality scores `Q` *can be transformed to miscall error probabilities with the R expression*

```
> Q <- 1 - 1 / (1 + 10^(-Q / 10))
```

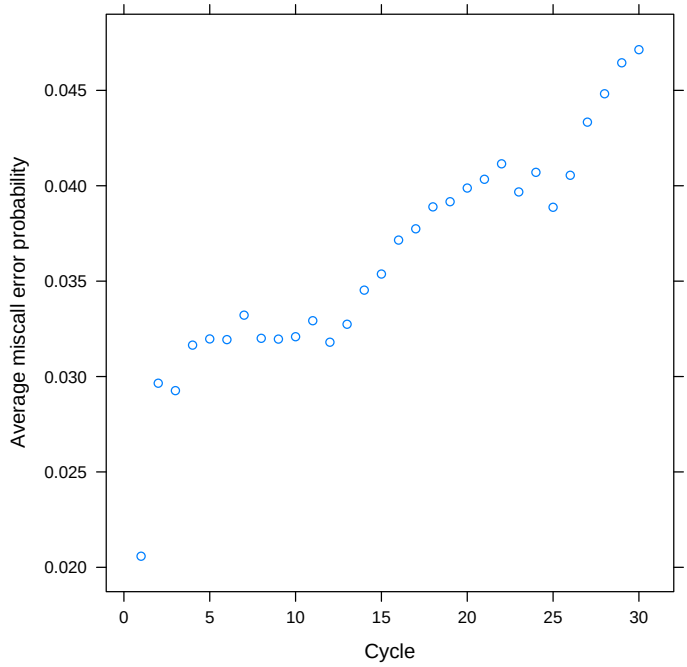
Evaluate this expression to transform the matrix of quality scores into a matrix of miscall probabilities.

Now use the function `colMeans` *to calculate the average quality at each cycle. If you get stuck, take a look at the solution below.*

Use the function `seq_along` *to create an R vector* `cycle` *containing the cycles, and the function* `xyplot` *to display the results as a simple graph. How does quality change with cycle?*

Note: Features in this plot reveal a variety of artifacts from the ELAND implementation, in addition to decisions about algorithm parameters made in generating this data.

```
> Q <- as(quality(aln), "matrix")[,1:30]
> Q <- 1 - 1 / (1 + 10^(-Q / 10))
> qByCycle <- colMeans(Q)
> cycle <- seq_along(qByCycle)
> xyplot(qByCycle ~ cycle,
+        ylab="Average miscall error probability", xlab="Cycle")
```

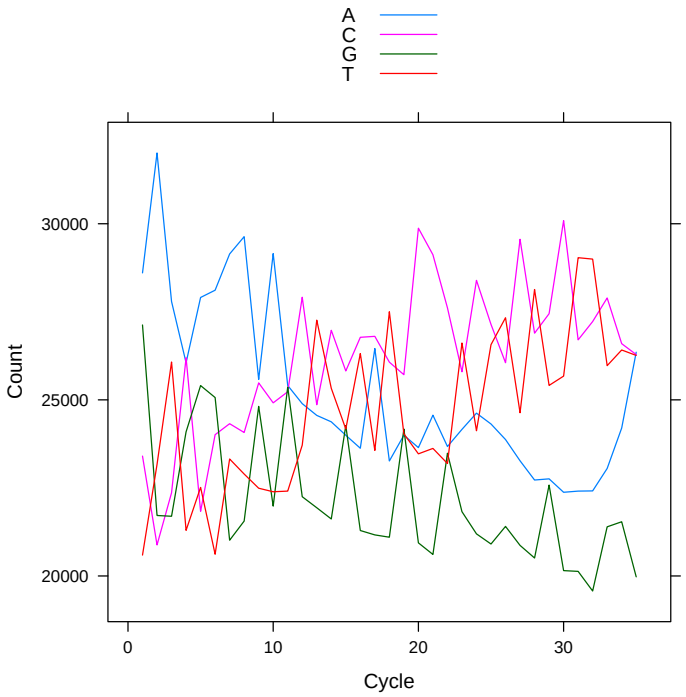


Exercise 12

As an advanced exercise, can you plot changes in nucleotide frequency along cycles of a read? The steps might involve:

1. *Use* `alphabetByCycle` *to calculate nucleotide counts per cycle.*
2. *'Flatten' the matrix into a data frame, with columns* `Nucleotide, Cycle, Count`*. The reason for creating a data frame like this is to ease use of the* `xyplot` *function, which expects data frames.*
3. *Use* `xyplot`*, including its* `group`*,* `auto.key` *and* `type` *arguments to plot the different* `Count` *as a function of* `Cycle`*, with points in the plot grouped by* `Nucleotide`*.*

```
> abc <- alphabetByCycle(sread(aln))[1:4,] # just nucs A, C, T, G
> df <- data.frame(Nucleotide=rownames(abc)[row(abc)],
+                  Cycle=as.vector(col(abc)),
+                  Count=as.vector(abc))
> print(xyplot(Count ~ Cycle, df, group=Nucleotide,
+              type="l",
+              auto.key=list(lines=TRUE, points=FALSE)))
```



4 Pattern matching / Pairwise alignment

In this section we will discuss the two types of pattern matching / pairwise alignment methodologies within the Biostrings package. We'll address the following general questions:

- How do the two different methodologies complement each other?
- Why does Biostrings contain yet another implementation of pattern matching / pairwise alignment?

4.1 Biostrings matching / alignment methodologies

Just like Solexa's support of ELAND and PhageAlign, Biostrings contains two different string matching implementations:

Exact and fuzzy string matching: The `matchPDict` family of functions allows the user to find all of the occurances of a set of patterns, with zero or a limited number of mismatches, within a target string.

Optimal pairwise alignment: The `pairwiseAlignment` function allows the user to fit three primary (global, local, overlap) and two derivative (subject overlap, pattern overlap) pairwise alignment types.

matchPDict	pairwiseAlignment
Utilizes a fast string matching algorithm.	Not practical for long strings.
Finds all occurrances with the with up to the specified # of mismatches.	Returns only the first occurrance of the best scoring alignment.
Supports removal of repeat masked regions.	Cannot handle masked genomes.
Produces limited output: # of times a pattern matches and where they occur.	Allows various summaries of alignments.
Does not support insertions or deletions.	Does support insertions and deletions.
Supports a limited number of matching options.	Provides a flexible alignment framework, including quality-based scoring.

Table 1: Comparisons of string matching/alignment methods.

For an overview of the these two functions, read the man pages by typing `help(matchPDict)` or `help(pairwiseAlignment)` at the R command line.

4.2 Matching against a short genome

High-throughput sequencing technologies like Solexa recommend using some sequencing cycles for quality assurance alignments against a simple genome. Solexa's SOP includes dedicating lane 5 from a set of 8 to sequencing the bacterophage ϕX174 genome, a circular single-stranded genome with 5386 the first to be sequenced back in 1978.

In order to provide an independent check of a Solexa run, we will map pre-aligned base calls to the phage genome. First we will read in the pre-aligned phage short reads using the `readBaseQuality` function from the ShortRead package.

```
> sp <- SolexaPath("MGED-2008/extdata/Solexa")
> srX174 <- readBaseQuality(sp,
+   seqPattern="s_5_.*_seq.txt", prbPattern="s_5_.*_prb.txt")
> alphabetFrequency(sread(srX174), collapse = TRUE)

     A      C      G      T      M      R      W      S
275752 244347 242849 317114      0      0      0      0
     Y      K      V      H      D      B      N      -
     0      0      0      0      0      0      0  10810
     +
     0
```

```
> csrX174 <- clean(srX174)
> csrX174

class: ShortReadQ
length: 29173 reads; width: 36 cycles
```

Now we will load in the Biostrings build-in `phiX174` object containing the sequence that is reported by Solexa to be used during the sequencing experiment. Given that the phage genome is circular, we will extend the 5' end with the first 35 bases of the 3' end.

```
> data(phiX174) # built-in data
> phiX174

  5386-letter "DNAString" instance
seq: GAGTTTTATCGCTTCCATGACGCAGA...AAAATGATTGGCGTATCCAACCTGCA

> phiX174Ext <- DNAString(paste(phiX174, phiX174[1:35], sep = ""))
> phiX174Ext

  5421-letter "DNAString" instance
seq: GAGTTTTATCGCTTCCATGACGCAGA...CGCTTCCATGACGCAGAAGTTAACAC
```

If all of the phage short reads align to the genome, then each location is expected to have a large coverage.

```
> nchar(sread(csrX174)[1]) * length(csrX174) / nchar(phiX174)

[1] 194.9922
```

For illustration purposes, we can generate all possible reads of the extended phage genome.

```
> psrX174 <-
+   list("F" =
+        views(phiX174Ext, 1:(nchar(phiX174Ext)-35), 36:nchar(phiX174Ext)),
+        "R" =
+        views(reverseComplement(phiX174Ext), 1:(nchar(phiX174Ext)-35),
+              36:nchar(phiX174Ext)))
> colMeans(alphabetFrequency(psrX174[["F"]], baseOnly = TRUE))

        A         C         G         T     other
 8.629038  7.733383  8.381730 11.255848  0.000000
```

The first type of matching we will use is exact string matching as implemented by the `matchPDict` function. This function requires a pattern dictionary (or *PDict*) as an input. Note that although the phage data is single stranded, the short reads were amplified with a PCR process that generated the reverse strand.

```
> epdictX174 <-
+   list("F" = PDict(sread(csrX174)),
+        "R" = PDict(reverseComplement(sread(csrX174))))
> ematchX174 <-
+   list("F" = matchPDict(epdictX174[["F"]], phiX174Ext),
+        "R" = matchPDict(epdictX174[["R"]], phiX174Ext))
> table(countIndex(ematchX174[["F"]]))

    0     1
20365  8808

> table(countIndex(ematchX174[["R"]]))

    0     1
22401  6772

> round(table(countIndex(ematchX174[["F"]]) +
+       countIndex(ematchX174[["R"]])) / length(csrX174), 3)

    0     1
0.466 0.534
```

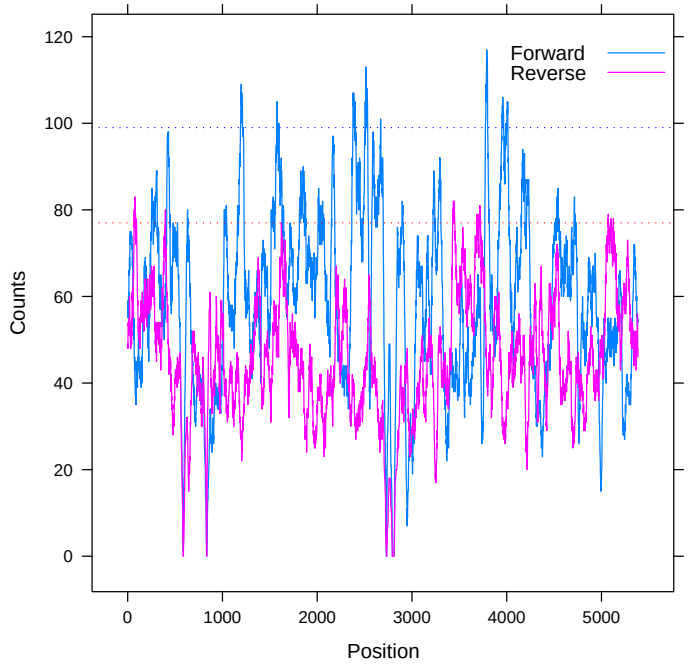
Exact matching to the genome only resulted in just over 50% alignment to the phage genome. We can expand the search to allow for up to 2 mismatches by specifying the `max.mismatch` argument as 2 in the `PDict` and `matchPDict` function calls (both are required).

```
> fpdictX174 <-
+   list("F" = PDict(sread(csrX174), max.mismatch = 2),
+        "R" = PDict(reverseComplement(sread(csrX174)), max.mismatch = 2))
> fmatchX174 <-
+   list("F" = matchPDict(fpdictX174[["F"]], phiX174Ext, max.mismatch = 2),
+        "R" = matchPDict(fpdictX174[["R"]], phiX174Ext, max.mismatch = 2))
> table(countIndex(fmatchX174[["F"]]))

    0     1     2
17571 11596     6

> table(countIndex(fmatchX174[["R"]]))

    0     1     2
20436  8731     6

> round(table(countIndex(fmatchX174[["F"]]) +
+       countIndex(fmatchX174[["R"]])) / length(csrX174), 3)

    0     1     2
0.303 0.697 0.000
```

In this case `matchPDict` function aligned about 70% of the phage short reads to the genome; 12 of which are mapped twice. We will leave further alignment of the unmatched phage short reads as an exercise.

Now we will turn our attention to where the phage short reads aligned to the genome by looking at the coverage.

```
> getCoverageX174 <- function(x) {
+     ans <- as.integer(coverage(x, 1, length(phiX174)))
+     ans[1:35] <- ans[1:35] +
+       as.integer(coverage(x, length(phiX174) + 1, length(phiX174Ext)))
+     ans
+ }
> coverageX174 <-
+   list("F" = getCoverageX174(ematchX174[["F"]]),
+        "R" = getCoverageX174(ematchX174[["R"]]))
> cor(coverageX174[["F"]], alphabetFrequency(psrX174[["F"]])[,"T"])

[1] 0.4618422

> peakbound <-
+   list("F" =
+        qpois(0.99,
+              nchar(sread(csrX174)[1]) * sum(countIndex(fmatchX174[["F"]]) != 0) /
+              nchar(phiX174)),
+        "R" =
+        qpois(0.99,
+              nchar(sread(csrX174)[1]) * sum(countIndex(fmatchX174[["R"]]) != 0) /
+              nchar(phiX174)))
> nChar <- length(coverageX174[["F"]])
> plotData <-
+   data.frame(Counts = c(coverageX174[["F"]], coverageX174[["R"]]),
+              Position = rep(1:nChar, 2),
+              Direction = rep(c("Forward", "Reverse"), each = nChar))
> print(xyplot(Counts ~ Position, groups = Direction, data = plotData,
+              panel =
+              function(...) {
+              panel.xyplot(...)
+              panel.abline(h = peakbound[["F"]], col = "blue", lty = 3)
+              panel.abline(h = peakbound[["R"]], col = "red", lty = 3)
+              },
+              type = "l",
+              auto.key =
+              list(x = 0.7, y = 0.95, points = FALSE, lines = TRUE)))
```



In order to find where there is a large accumulations of matches, we can use the `slice` to obtain the large peaks.

```
> slice(coverageX174[["F"]], lower = peakbound[["F"]])

  Views on a 5386-integer XInteger subject
subject: 56 56 58 59 55 56 56 60 60 61 60 61 66 65 66 65 65 68 ...
views:
     start  end width
 [1]  1190 1212    23 [ 99  99  99 103 103 107 105 107 ...]
 [2]  1214 1216     3 [99 99 99]
 [3]  1218 1219     2 [99 99]
 [4]  1573 1580     8 [101 102 105 104 103 103 102  99]
 [5]  1586 1586     1 [99]
 [6]  1588 1588     1 [99]
 [7]  1594 1596     3 [100 100 100]
 [8]  2377 2395    19 [101 106 107 106 105 104 105 105 ...]
 [9]  2398 2403     6 [102 102 105 105 101  99]
[10]  2506 2509     4 [ 99 105 102 100]
[11]  2511 2532    22 [101 103 103 110 113 112 109 111 ...]
[12]  2669 2670     2 [101  99]
[13]  3775 3798    24 [ 99 100 101 101 101 105 108 109 ...]
[14]  3949 3968    20 [ 99 100 102 101 102 104 103 100 ...]
```

```
[15]  3992 3993     2 [ 99 100]
[16]  3998 3998     1 [99]
[17]  4001 4012    12 [100 102 103 103 102 105 105 105 ...]
```

One interesting feature of the coverage for the phage short reads is that there are a handful of positions with very low counts. We can examine these locations with the `pairwiseAlignment` function.

```
> phiX174Substr1 <- substring(phiX174, 587-35, 587+35)
> phiX174Substr2 <- substring(phiX174, 833-35, 833+35)
> phiX174Substr3 <- substring(phiX174, 2731-35, 2731+35)
> phiX174Substr4 <- substring(phiX174, 2793-35, 2811+35)
> subalign4 <-
+   pairwiseAlignment(sread(csrX174), phiX174Substr4,
+                     type = "subjectOverlap")
> summary(subalign4[score(subalign4) > 0])

Subject Overlap Pairwise Alignment
Number of Alignments:  275

Scores:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1942 31.9400 55.5800 45.9500 63.4600 63.4600

Number of matches:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  26.00   31.00   34.00   32.83   35.00   35.00

Top 10 Mismatch Counts:
    SubjectPosition Subject Pattern Count Probability
84               54       C       T   136  0.95774648
48               36       C       T   130  0.97744361
131              78       G       T     9  0.09090909
119              70       A       T     8  0.06666667
80               53       A       C     8  0.05517241
116              69       C       T     7  0.05785124
79               52       A       T     7  0.04861111
118              70       A       G     6  0.05000000
73               50       A       C     6  0.04347826
34               29       T       G     6  0.04255319
```

It appears that the reference genome used by Solexa differs from the actual sequence. Verifying these errors is left as an exercise.

```
> revisedPhiX174 <-
+   replaceLetterAtLoc(phiX174, c(587,833,2731,2793,2811), "AAGTT")
> revisedPhiX174Ext <-
```

```
+   replaceLetterAtLoc(phiX174Ext, c(587,833,2731,2793,2811), "AAGTT")
> table(countPDict(epdictX174[["F"]], revisedPhiX174Ext))

    0     1
20041  9132

> table(countPDict(epdictX174[["R"]], revisedPhiX174Ext))

    0     1
22178  6995
```

The last topics we will address in this subsection is refining the quality scores associated with the short reads first by the preliminary quality scores assigned by Solexa and second by considering the quality associated with each cycle of the reads. More complicated determinations of quality are left to the reader.

```
> set.seed(123)
> srsamp <- csrX174[sample(length(csrX174), 2000)]
> alignq <-
+   list("F" =
+        pairwiseAlignment(sread(srsamp), revisedPhiX174Ext,
+                          patternQuality =
+                          SolexaQuality(quality(quality(srsamp))),
+                          subjectQuality = SolexaQuality(99L),
+                          type = "subjectOverlap"),
+        "R" =
+        pairwiseAlignment(reverseComplement(sread(srsamp)),
+                          revisedPhiX174Ext,
+                          patternQuality =
+                          SolexaQuality(quality(quality(srsamp))),
+                          subjectQuality = SolexaQuality(99L),
+                          type = "subjectOverlap"))
> cutoffq <- max(pmin(score(alignq[["F"]]), score(alignq[["R"]])))
> missumq <-
+   list("F" = mismatchSummary(alignq[["F"]][score(alignq[["F"]]) > cutoffq]),
+        "R" = mismatchSummary(alignq[["R"]][score(alignq[["R"]]) > cutoffq]))
> denom <-
+   alphabetFrequency(quality(quality(srsamp)[
+     pmax(score(alignq[["F"]]), score(alignq[["R"]])) > cutoffq]),
+       collapse = TRUE)
> denom <- denom[denom > 0]
> old <-
+   SolexaQuality(paste(rownames(missumq[["F"]]$pattern$quality), collapse = ""))
> qualerrq <-
+   (missumq[["F"]]$pattern$quality[,"Count"] +
+    missumq[["R"]]$pattern$quality[,"Count"]) / denom
> new <- SolexaQuality(rev(cummax(rev(qualerrq))))
> as.integer(old)
```

```
 [1] -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13
[19] 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
[37] 32 33 34 35 36 37 38 39 40
```

```
> as.integer(new)
```

```
 [1]  3  3  3  3  3  3  5  5  6  7  8  8  8 10 10 11 11 11
[19] 11 11 13 13 14 14 14 14 15 15 15 15 15 16 16 16 16 16
[37] 18 18 18 18 18 18 18 18 21
```

```
> newquality <-
+   chartr(as.character(old), as.character(new), quality(quality(srsamp)))
> newquality[[1]]
```

```
  36-letter "BString" instance
seq: UUUKJURKJHKUHCENHUUKMUPJNUMPUNEUCPOG
```

Determining the marginal position based quality measures are much easier to obtain.

```
> alignc <-
+   list("F" =
+        pairwiseAlignment(sread(srsamp), revisedPhiX174Ext,
+                          type = "subjectOverlap"),
+        "R" =
+         pairwiseAlignment(reverseComplement(sread(srsamp)),
+                           revisedPhiX174Ext,
+                           type = "subjectOverlap"))
> cutoffc <- max(pmin(score(alignc[["F"]]), score(alignc[["R"]])))
> qualerrc <-
+   (mismatchSummary(
+     alignc[["F"]][score(alignc[["F"]]) > cutoffc])$pattern$position[,"Count"] +
+    rev(mismatchSummary(
+        alignc[["R"]][score(alignc[["R"]]) > cutoffc])$pattern$position[,"Count"])) /
+   sum(pmax(score(alignc[["F"]]), score(alignc[["R"]])) > cutoffc)
> SolexaQuality(qualerrc)
```

```
  A SolexaQuality instance of length 1
    width seq
[1]    36 QOQRPPPPOPPPQPPPOPPPOOONNNOOONNNMMLL
```

Exercise 13

1. *Use the* `pairwiseAlignment` *function to generate the forward and reverse alignment scores for the phage short reads that have more than two mismatches. (This may take a few minutes). How many additional alignments can you find?*

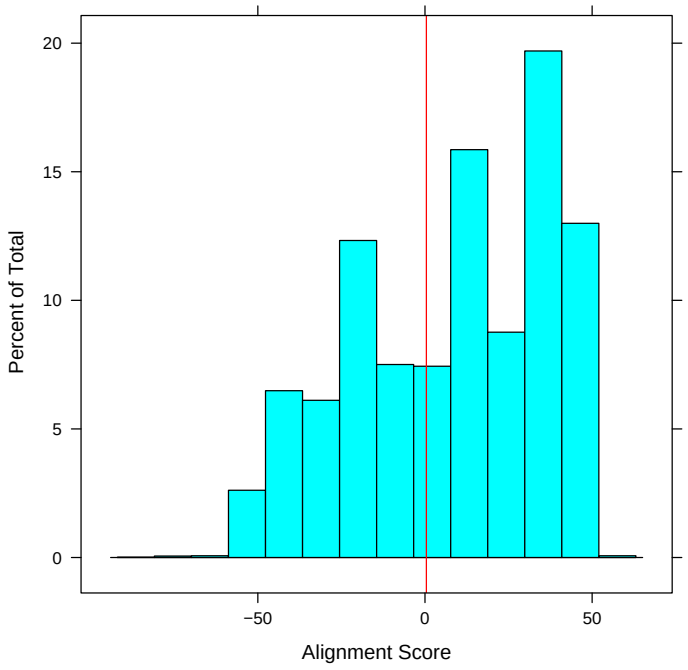
2. *Verify that the changes made to the phage genome are justifiable.*

1. Solution 1

```
> unalign <- (countIndex(fmatchX174[["F"]]) + countIndex(fmatchX174[["R"]]) == 0)
> palignX174 <-
+   list("F" =
+        pairwiseAlignment(sread(csrX174)[unalign], phiX174Ext,
+                          type = "subjectOverlap", scoreOnly = TRUE),
+        "R" =
+        pairwiseAlignment(reverseComplement(sread(csrX174)[unalign]), phiX174Ext,
+                          type = "subjectOverlap", scoreOnly = TRUE))
> cutoffex <- max(pmin(palignX174[["F"]], palignX174[["R"]]))
> table(pmax(palignX174[["F"]], palignX174[["R"]]) > cutoffex)
```

```
FALSE  TRUE
 3192  5642
```

```
> print(histogram(~ pmax(palignX174[["F"]], palignX174[["R"]]),
+   panel = function(...) {panel.histogram(...); panel.abline(v=cutoffex, col = "red")}
+   xlab = "Alignment Score"))
```



2. Solution 2

```
> for(i in 1:4) {
+   objName <- paste("subalign", i, sep = "")
```

```
+   assign(objName,
+          pairwiseAlignment(sread(csrX174), get(paste("phiX174Substr", i, sep ="")),
+                            type = "subjectOverlap"))
+   print(summary(get(objName)[score(get(objName)) > 0]))
+ }
```

4.3 Matching against a large genome

The `matchPDict` function is on par with commonly used alignment software like ELAND and MAQ in terms of performance, but much more flexible in terms of application. We will use `matchPDict` to align short reads against the mouse genome. First weeread in the ELAND aligned reads.

```
> sp <- SolexaPath("MGED-2008/extdata/Solexa")
> aln <- readAligned(sp, pattern="s_1_export.txt")
> caln <- clean(aln)
> caln

class: AlignedRead
length: 96818 reads; width: 35 cycles
chromosome: chrUn_random.fa chr1.fa ... 255:255:255 NM
position: 5251632 163068613 ... NA NA
strand: F R ...
alignQuality: NumericQuality
alignData varLabels: run lane ... y filtering
```

Next we will create pattern dictionaries for the forward and reverse strands. To bring the results on par with ELAND, we will allow for up to two mismatches.

```
> alndict <-
+   list("F" = PDict(sread(caln), max.mismatch = 2),
+        "R" = PDict(reverseComplement(sread(caln)), max.mismatch = 2))
```

Then we will load the *Mus musculus* genome provided by the UCSC. This genome, by default, is "masked" to hide information like assembly gaps and RepeatMasker identified sequences. We will turn off two of the three masks and leave the unmasking of the third for an exercise.

```
> library(BSgenome.Mmusculus.UCSC.mm9)
> Mmusculus

Mouse genome
|
| organism: Mus musculus
| provider: UCSC
| provider version: mm9
```

```
| release date: Jul. 2007
| release name: NCBI Build 37
|
| single sequences (see '?seqnames'):
|   chr1          chr2          chr3          chr4
|   chr5          chr6          chr7          chr8
|   chr9          chr10         chr11         chr12
|   chr13         chr14         chr15         chr16
|   chr17         chr18         chr19         chrX
|   chrY          chrM          chr1_random   chr3_random
|   chr4_random   chr5_random   chr7_random   chr8_random
|   chr9_random   chr13_random  chr16_random  chr17_random
|   chrX_random   chrY_random   chrUn_random
|
| multiple sequences (see '?mseqnames'):
|   upstream1000  upstream2000  upstream5000
|
| (use the '$' or '[[' operator to access a given sequence)

> chr1 <- Mmusculus$chr1
> masks(chr1)

  A MaskCollection instance of length 3 and width 197195432
masks:
  maskedwidth maskedratio active
1     5717956  0.02899639   TRUE
2    84650265  0.42927092   TRUE
3     4014755  0.02035927   TRUE
                              names
1                      assembly gaps
2                       RepeatMasker
3 Tandem Repeats Finder [period<=12]
all masks together:
  maskedwidth maskedratio
     90481595   0.4588422

> active(masks(chr1))[1] <- FALSE
> active(masks(chr1))[3] <- FALSE
> masks(chr1)

  A MaskCollection instance of length 3 and width 197195432
masks:
  maskedwidth maskedratio active
1     5717956  0.02899639  FALSE
2    84650265  0.42927092   TRUE
3     4014755  0.02035927  FALSE
                              names
```

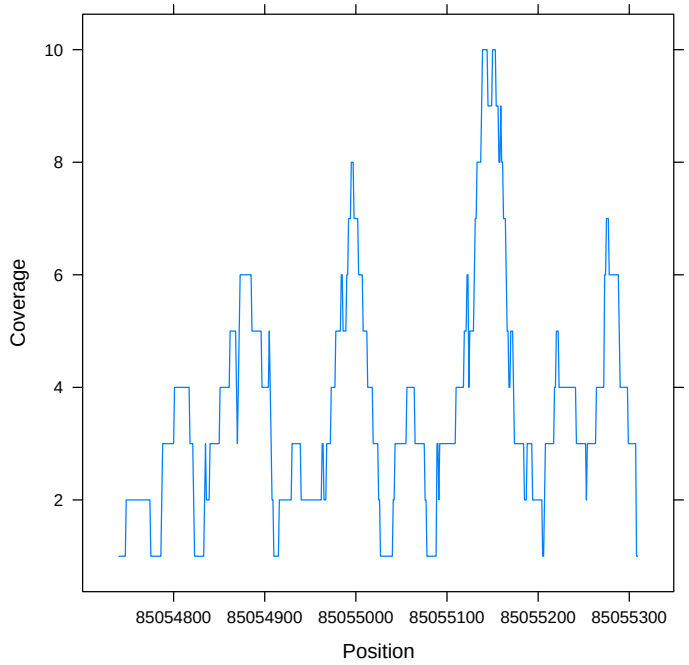
```
1                      assembly gaps
2                       RepeatMasker
3 Tandem Repeats Finder [period<=12]
all masks together:
  maskedwidth maskedratio
     90481595   0.4588422
all active masks together:
  maskedwidth maskedratio
     84650265   0.4292709
```

Now we can align the short reads against chromosome 1.

```
> chr1aln <-
+   list("F" = matchPDict(alndict[["F"]], chr1, max.mismatch = 2),
+        "R" = matchPDict(alndict[["R"]], chr1, max.mismatch = 2))
> table(countIndex(chr1aln[["F"]]) + countIndex(chr1aln[["R"]]))

    0     1     2     3     4     5     6     7     8     9
93969  2331    81    45    36    60    48    77   100    52
   10    11    13    22    27    35    39    45   100   116
    7     2     1     1     1     2     1     1     1     2

> table(chromosome(caln) == "chr1.fa",
+       countIndex(chr1aln[["F"]]) + countIndex(chr1aln[["R"]]) > 0)

        FALSE  TRUE
  FALSE 92637   710
  TRUE   1332  2139
```

With these alignments, we can plot a short segment of the coverage for chromosome 1.

```
> chr1reduce <- list("F" = reduce(unlist(chr1aln[["F"]])),
+                    "R" = reduce(unlist(chr1aln[["R"]])))
> tail(sort(width(chr1reduce[["F"]])))

[1] 338 338 338 371 521 570

> crange <-
+   chr1reduce[["F"]][width(chr1reduce[["F"]]) == max(width(chr1reduce[["F"]]))]
> plotData <- data.frame(Position = start(crange):end(crange),
+   Coverage = as.integer(coverage(chr1aln[["F"]], start(crange), end(crange))))
> print(xyplot(Coverage ~ Position, data = plotData, type = "l"))
```



We can use `pairwiseAlignment` to further probe the region of chromosome 1 selected above.

```
> chr1substr <-
+   DNAString(as.character(views(Mmusculus[["chr1"]], start(crange), end(crange))))
> pscores <-
+   list("F" =
+        pairwiseAlignment(sread(caln), chr1substr, type = "subjectOverlap",
+                          scoreOnly = TRUE),
+        "R" =
+        pairwiseAlignment(reverseComplement(sread(caln)), chr1substr,
+                          type = "subjectOverlap", scoreOnly = TRUE))
> qualityMatrices <- qualitySubstitutionMatrices()
> cutoffpaln <-
+   34.5 * qualityMatrices[["match"]]["22", "22"] +
+     1.5 * qualityMatrices[["mismatch"]]["22", "22"]
> chr1palign <-
+   pairwiseAlignment(sread(caln)[pscores[["F"]] >= cutoffpaln], chr1substr,
+                     type = "subjectOverlap")
> table(nmismatch(chr1palign))

 0  1
12 27
```

```
> mismatchSummary(chr1palign)$subject

   SubjectPosition Subject Pattern Count Probability
1               45       C       T     1   1.0000000
2               77       T       C     1   0.5000000
3              123       C       T     1   0.2000000
4              182       A       G     2   1.0000000
5              227       C       T     1   1.0000000
6              256       A       T     1   0.1428571
7              260       G       T     1   0.1666667
8              272       G       A     4   1.0000000
9              324       C       T     1   0.5000000
10             370       T       C     1   1.0000000
11             398       C       A     1   0.2500000
12             410       A       T     2   0.3333333
13             447       T       A     2   1.0000000
14             491       C       T     4   1.0000000
15             511       T       C     1   0.3333333
16             536       G       C     1   0.2500000
17             551       G       A     2   1.0000000
```

Exercise 14

1. *Rerun the alignment against chromosome 1 without any masks. (This may take five minutes or so).*

```
> chr1 <- Mmusculus[["chr1"]
> masks(chr1)
> active(masks(chr1)) <- FALSE
> masks(chr1)
> unmaskchr1aln <-
+   list("F" = matchPDict(alndict[["F"]], chr1, max.mismatch = 2),
+        "R" = matchPDict(alndict[["R"]], chr1, max.mismatch = 2))
> table(countIndex(unmaskchr1aln[["F"]]) + countIndex(unmaskchr1aln[["R"]]) > 0)

FALSE  TRUE
86935  9883

> table(chromosome(caln) == "chr1.fa",
+       countIndex(unmaskchr1aln[["F"]]) + countIndex(unmaskchr1aln[["R"]]) > 0)

       FALSE  TRUE
  FALSE 86800  6547
  TRUE    135  3336
```

5 Advanced topics

ShortRead and Biostrings provide a great deal of flexiblity. We'll walk through the following demos, time permitting.

- Remapping probes. Biostrings can be used to easily remap probes; the operation is fast enough to be included in an example:

  ```
  > example(matchPDict)
  ```

- Biostrings includes facilities to 'mask' regions of the genome so that they are not searched. This can be computationally efficient (e.g., masked regions do not need to be searched) and biologically relevant (e.g., unmasked regions may represent targes of a ChIP-seq experiment).

- SNPs can be 'injected' into the genome as their IUPAC code. This allows ambiguous matching on the subject sequence.

- R's flexible graphics capabilities can display sequence-scale information in novel and informative ways. For instance, the R package HilbertDisplayCurve presents the genome in a compact two-dimensional representation that clearly discriminates, e.g., diffuse regions of methylation versus highly specific regions.

6 Resources

Internet

- Home page: `http://bioconductor.org`
- Mailing lists: `http://bioconductor.org/docs/mailList.html`. `bioc` for general help and informed discussion, `bioc-sig-sequencing` for short read and other sequencing related topics.
- BioConductor installation instructions: `http://bioconductor.org/install`.
- Pages with links to package vignettes:
  - `http://bioconductor.org/packages/devel/bioc/html/Biostrings.html`
  - `http://bioconductor.org/packages/devel/bioc/html/ShortRead.html`

Creating this document:

- R version 2.8.0 Under development (unstable) (2008-08-22 r46416), `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_PAPER=en_US.UTF-8; 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, tools, utils
- Other packages: BSgenome 1.9.9, BSgenome.Mmusculus.UCSC.mm9 1.3.7, Biobase 2.1.3, Biostrings 2.9.66, IRanges 0.99.7, ShortRead 0.1.49, lattice 0.17-8
- Loaded via a namespace (and not attached): grid 2.8.0