Practical: Working with Sequence Data

Martin Morgan (mtmorgan@fhcrc.org)

November 18-22, 2013

Contents

Common file formats The 'big data' component of high-throughput sequence analyses seems to be a tangle of transformations between file types; common files are summarized in Table˜1. FASTQ and BAM (sometimes CRAM) files are the primary formats for representing raw sequences and their alignments. VCF are used to summarize called variants in DNA-seq; BED and sometimes WIG files are used to represent ChIP and other regulatory peaks and 'coverage'. GTF / GFF files are important for providing feature annotations, e.g., of exons organization into transcripts and genes.

1 Short reads: FASTQ files

Table 1: Common file types and *Bioconductor* packages used for input.

File	Description	Package
FASTQ	Unaligned sequences: identifier, sequence, and encoded quality score tuples	*ShortRead*
BAM	Aligned sequences: identifier, sequence, reference sequence name, strand position, cigar and additional tags	*Rsamtools*
VCF	Called single nucleotide, indel, copy number, and structural variants, often compressed and indexed (with *Rsamtools* `bgzip`, `indexTabix`)	*VariantAnnotation*
GFF, GTF	Gene annotations: reference sequence name, data source, feature type, start and end positions, strand, etc.	*rtracklayer*
BED	Range-based annotation: reference sequence name, start, end coordinates.	*rtracklayer*
WIG, bigWig	'Continuous' single-nucleotide annotation.	*rtracklayer*
2bit	Compressed FASTA files with 'masks'	

1.1 FASTQ files

The Illumina GAII and HiSeq technologies generate sequences by measuring incorporation of florescent nucleotides over successive PCR cycles. These sequencers produce output in a variety of formats, but *FASTQ* is ubiquitous. Each read is represented by a record of four components:

```
## @ERR127302.200 HWI-EAS350_0441:1:1:1196:1175#0/2
## CTCAGCGACACCAATCTCCTCCTTGAAGTGCACGTGGACCAGGCCCTCCGCCCGACGGCGGGAGTTGGGGAG
## +
## B<BGGADEAB>73A/(?BD8EBAAA<?>>>BAAA><+*<8:,-5&E>??;B#####################
```

The first and third lines (beginning with `@` and `+` respectively) are unique identifiers. The identifier produced by the sequencer typically includes a machine id followed by colon-separated information on the lane, tile, x, and y coordinate of the read. The example illustrated here also includes the SRA accession number, added when the data was submitted to the archive. The machine identifier could potentially be used to extract information about batch effects. The spatial coordinates (lane, tile, x, y) are often used to identify optical duplicates; spatial coordinates can also be used during quality assessment to identify artifacts of sequencing, e.g., uneven amplification across the flow cell, though these spatial effects are rarely pursued.

The second and fourth lines of the FASTQ record are the nucleotides and qualities of each cycle in the read. This information is given in 5' to 3' orientation as seen by the sequencer. A letter `N` in the sequence is used to signify bases that the sequencer was not able to call. The fourth line of the FASTQ record encodes the quality (confidence) of the corresponding base call. The quality score is encoded following one of several conventions, with the general notion being that letters later in the visible ASCII alphabet

```
##  !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
##  ?  @  A  B  C  D  E  F  G  H  I  J
## 30 31 32 33 34 35 36 37 38 39 40 41
```

are of higher quality. Letters map to numbers, and numbers correspond (most commonly) to $-10 \log_{10} p$. In the encoding above, `I` corresponds to a phred score of 40, hence $p = 0.0001$. Both the sequence and quality scores may span multiple lines.

1.2 Quality assessment

Exercise 1 *Here we create a quality assessment report of FASTQ files from the ArrayExpress repository E-MTAB-1147[1] [1].*

a. Load the ShortRead and parallel packages, and enable parallel evaluation of ShortRead functions with `options(srapply_fapply="parallel")`.
b. Create a character vector `dirPath` to the 'bigdata/fastq' directory containing the files 'ERR127302_1.fastq.gz', 'ERR127302_2.fastq.gz'.
c. Create a quality report for these two files using the `ShortRead::qa` function, e.g., `qa <- qa(dirPath, "ERR*", type="fastq")`.
d. View the quality report in a web browser with `browseURL(report(qa))`
e. View the QA report for all fastq files in the full experiment. Do this by loading the prepared data object 'E-MTAB-1147-qa_report.Rda'. Discuss the meaning of each plot with your neighbor. What technology arti

Solution: Load the *ShortRead* and *parallel* packages

```
library(ShortRead)
library(parallel)
options(srapply_fapply="parallel")
```

[1] http://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-1147/

Create the QA report from two sample files

```
dirPath <- "~/bigdata/fastq"
qa <- qa(dirPath, "ERR*", type="fastq")
```

View the report

```
browseURL(report(qa))
```

Load the report for all lanes

```
(load(file.path(dirPath, "E-MTAB-1147-qa_report.Rda")))

## [1] "qa"
```

View the report

```
browseURL(report(qa))
```

1.3 Trimming

Exercise 2 *This exercise explores trimming, then applies a trimming filter to several FASTQ files.*

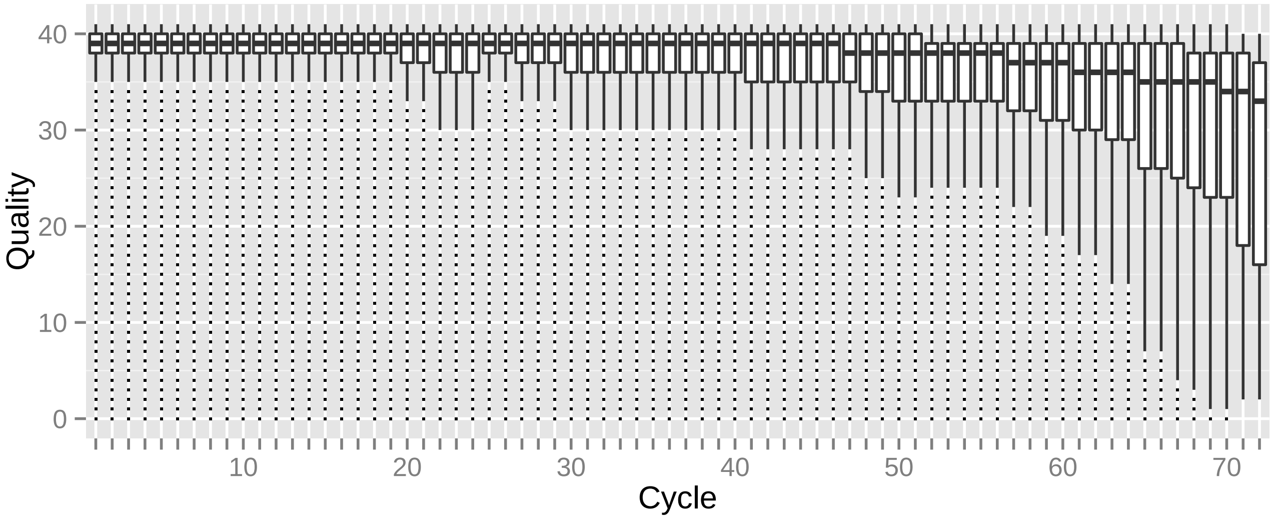
Be sure to start by loading the EMBOBGI package.

```
library(EMBOBGI)
```

a. *Create a character vector pointing to a FASTQ file,* `fl <- file.path(dirPath, "ERR127302_1.fastq.gz")`
b. *Load a random sample of 100,000 reads from the FASTQ file, using* `fq <- FastqSampler()` *and* `srq <- yield(fq)`
c. *Plot a histogram of qualities as a function of cycle,* `plotByCycle(srq)`*.*
d. *Look at how qualities are encoded using* `encoding(quality(srq))`*.*
e. *Trim reads after the first 3 nucleotides with aveage quality less than 20 (encoding* `"5"`*) using* `trimTails(srq, 3, "5")`

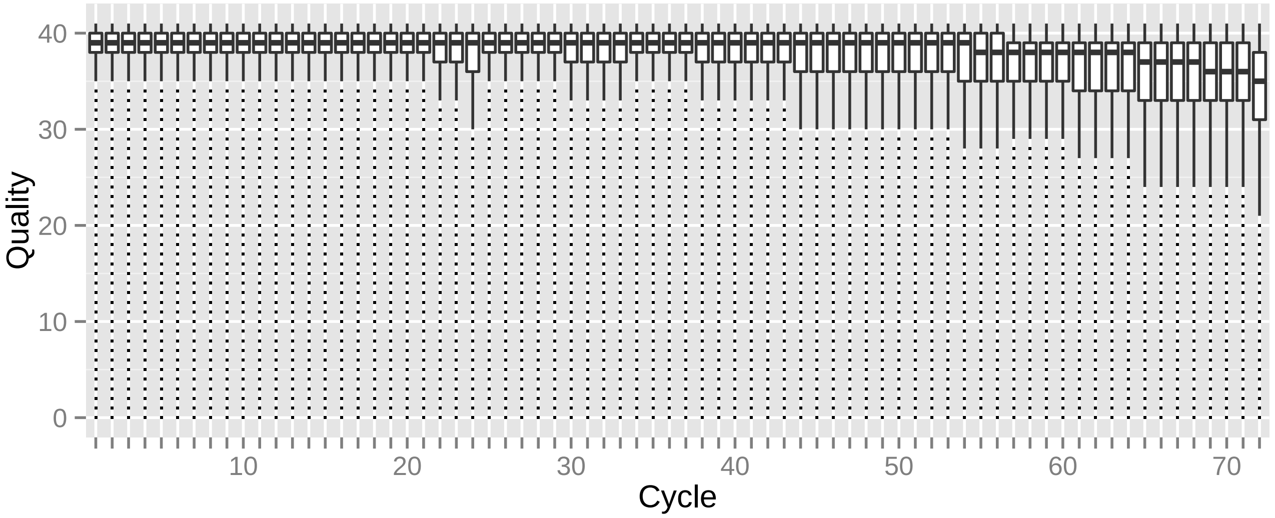
Solution: Load a sample of 100,000 reads and visualize their quality

```
fl <- file.path(dirPath, "ERR127302_1.fastq.gz")
fq <- FastqSampler(fl, 100000)
srq <- yield(fq)
srq

## class: ShortReadQ
## length: 100000 reads; width: 72 cycles

plotByCycle(quality(srq))
```



Trim the reads and visualize qualities

```
trimmed <- trimTails(srq, 3, "5")
plotByCycle(quality(trimmed))
```



Exercise 3 *(Optional) This exercise trims all reads using our trimming function.*

a. *List the full path to all fastq files using `fls <- dir(dirPath, pattern="fastq.gz", full=TRUE)`.*
b. *Create destination files for each source file, using `destinations <- sub("fastq.gz", "trimmed.fastq", fls)`*
c. *Trim reads as before, but use the file paths and destinations as arguments, `trimTails(fls, 3, "5", FALSE, destinations=destinations)`.*

Solution: Identify relevant files

```
(fls <- dir(dirPath, pattern="fastq.gz", full=TRUE))

## [1] "/home/mtmorgan/bigdata/fastq/ERR127302_1.fastq.gz"
## [2] "/home/mtmorgan/bigdata/fastq/ERR127302_2.fastq.gz"
```

Map the file names to destinations

```
(destinations <- sub("fastq.gz", "trimmed.fastq", fls))

## [1] "/home/mtmorgan/bigdata/fastq/ERR127302_1.trimmed.fastq"
## [2] "/home/mtmorgan/bigdata/fastq/ERR127302_2.trimmed.fastq"
```

Perform the trimming

```
trimTails(fls, 2, "5", destinations=destinations)
```

2 Aligned reads: BAM files

2.1 BAM files

Most down-stream analysis of short read sequences is based on reads aligned to reference genomes. There are many aligners available, including BWA [2, 3], Bowtie / Bowtie2 [4], and GSNAP; merits of these are discussed in the literature. There are also alignment algorithms implemented in *Bioconductor* (e.g., `matchPDict` in the *Biostrings* package, and the *Rsubread* package); `matchPDict` is particularly useful for flexible alignment of moderately sized subsets of data.

Alignment formats Most main-stream aligners produce output in SAM (text-based) or BAM format. A SAM file is a text file, with one line per aligned read, and fields separated by tabs. Here is an example of a single SAM line, split into fields.

```
fl <- system.file("extdata", "ex1.sam", package="Rsamtools")
strsplit(readLines(fl, 1), "\t")[[1]]

##  [1] "B7_591:4:96:693:509"                  "73"
##  [3] "seq1"                                 "1"
##  [5] "99"                                   "36M"
##  [7] "*"                                    "0"
##  [9] "0"                                    "CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG"
## [11] "<<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<;:<;7" "MF:i:18"
## [13] "Aq:i:73"                              "NM:i:0"
## [15] "UQ:i:0"                               "H0:i:1"
## [17] "H1:i:0"
```

Fields in a SAM file are summarized in Table˜2. We recognize from the FASTQ file the identifier string, read sequence and quality. The alignment is to a chromosome 'seq1' starting at position 1. The strand of alignment is encoded in the 'flag' field. The alignment record also includes a measure of mapping quality, and a CIGAR string describing the nature of the alignment. In this case, the CIGAR is 36M, indicating that the alignment consisted of 36 `M`atches or mismatches, with no indels or gaps; indels are represented by `I` and `D`; gaps (e.g., from alignments spanning introns) by `N`.

BAM files encode the same information as SAM files, but in a format that is more efficiently parsed by software; BAM files are the primary way in which aligned reads are imported in to *R*.

Table 2: Fields in a SAM record. From http://samtools.sourceforge.net/samtools.shtml

Field	Name	Value
1	QNAME	Query (read) NAME
2	FLAG	Bitwise FLAG, e.g., strand of alignment
3	RNAME	Reference sequence NAME
4	POS	1-based leftmost POSition of sequence
5	MAPQ	MAPping Quality (Phred-scaled)
6	CIGAR	Extended CIGAR string
7	MRNM	Mate Reference sequence NaMe
8	MPOS	1-based Mate POSition
9	ISIZE	Inferred insert SIZE
10	SEQ	Query SEQuence on the reference strand
11	QUAL	Query QUALity
12+	OPT	OPTional fields, format TAG:VTYPE:VALUE

2.2 Gapped alignments in *R*

The readGAlignments function from the *GenomicRanges* package reads essential information from a BAM file in to *R*. The result is an instance of the *GappedAlignments* class. The *GappedAlignments* class has been designed to allow useful manipulation of many reads (e.g., 20 million) under moderate memory requirements (e.g., 4 GB).

Exercise 4 *This exercise explores the GappedAlignments class.*

a. *Load the RNAseqData.HNRNPC.bam.chr14 and retrieve the names of the BAM files it contains. These BAM files are subsets of a larger experiment.*
b. *Read one BAM file in to R using readGAlignments. How many reads are there? What do the first few records look like?*
c. *Use the strand accessor and the standard R function table to tabulate the number of reads on the plus and minus strand. Use the width and cigar accessors to summarize the aligned width and to explore the alignment cigars.*
d. *The readGAlignments function takes an additional argument, param, allowing the user to specify regions of the BAM file (e.g., known gene coordinates) from which to extract alignments, and other data to be extracted from the BAM file. Create a ScanBamParam object with argument what="seq", and use this to input the read sequences as well as basic alignment information.*
e. *With larger BAM files we often want to iterate through the file in chunks. Do this by creating a BamFile from a file path, specifying a yieldSize. Then write a short loop that uses readGAlignments to input successive chunks until there are no more records left.*

Solution: Load the experiment data library and read in one file, discovering the number of reads present

```
library(RNAseqData.HNRNPC.bam.chr14)
fls <- RNAseqData.HNRNPC.bam.chr14_BAMFILES
basename(fls)

## [1] "ERR127306_chr14.bam" "ERR127307_chr14.bam" "ERR127308_chr14.bam"
## [4] "ERR127309_chr14.bam" "ERR127302_chr14.bam" "ERR127303_chr14.bam"
## [7] "ERR127304_chr14.bam" "ERR127305_chr14.bam"

aln <- readGAlignments(fls[1])
length(aln)

## [1] 800484

head(aln, 3)
```

```
## GAlignments with 3 alignments and 0 metadata columns:
##       seqnames strand       cigar    qwidth     start       end     width      ngap
##          <Rle>  <Rle> <character> <integer> <integer> <integer> <integer> <integer>
##   [1]    chr14      +         72M        72  19069583  19069654        72         0
##   [2]    chr14      +         72M        72  19363738  19363809        72         0
##   [3]    chr14      -         72M        72  19363755  19363826        72         0
##   ---
##   seqlengths:
##                       chr1                 chr10 ...                  chrY
##                  249250621             135534747 ...              59373566
```

A *GappedAlignments* instance is like a data frame, but with accessors as suggested by the column names. It is easy to query, e.g., the distribution of reads aligning to each strand, the width of reads, or the cigar strings

```
table(strand(aln))

##
##      +      -      *
## 400242 400242      0

range(width(aln))

## [1]     70 404751

head(sort(table(cigar(aln)), decreasing=TRUE))

##
##        72M 35M123N37M 38M670N34M  64M316N8M 36M123N36M 18M123N54M
##     603939        272        264        261        228        225
```

Here we construct a *ScanBamParam* object and indicate that we would also like to input the read sequence.

```
param <- ScanBamParam(what="seq")
aln <- readGAlignments(fls[1], param=param)
```

To iterate through a BAM file, create a *BamFile* instance with appropriate `yieldSize`. We use `yieldSize=200000` in the work below, but in reality this could be one or two orders of magnitude larger.

```
bf <- open(BamFile(fls[1], yieldSize=200000))
repeat {
    aln <- readGAlignments(bf)
    if (length(aln) == 0)
        break                           # no more records
    ## do work
    message(length(aln))
}

## 200000
## 200000
## 200000
## 200000
## 484

close(bf)
```

2.3 Summarizing overlaps

Exercise 5 *A basic operation in RNA-seq and other work flows is to count the number of times aligned reads overlap features of interest.*

a. *Load the 'transcript db' package that contains the coordinates of each exon of the UCSC 'known genes' track of hg19.*
b. *Extract the exon coordinates grouped by gene; the result is an GRangesList object that we will discuss more latter.*
c. *Use the `summarizeOverlaps` function with the exon coordinates and BAM files to generate a count of the number of reads overlapping each gene. Visit the help page `?summarizeOverlaps` to read about the counting strategy used.*
d. *The counts can be extracted from the return value of `summarizeOverlaps` using the function `assay`. This is standard R matrix. How many reads overlapped regions of interest in each sample? How many genes had non-zero counts?*

Solution:

```
## library(parallel); options(mc.cores=detectCores())
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
ex <- exonsBy(TxDb.Hsapiens.UCSC.hg19.knownGene, "gene")
counts <- summarizeOverlaps(ex, fls)
colSums(assay(counts))

## ERR127306 ERR127307 ERR127308 ERR127309 ERR127302 ERR127303 ERR127304 ERR127305
##    340669    373302    371666    331540    313817    331160    331639    329672

sum(rowSums(assay(counts)) != 0)

## [1] 528
```

3 Variants: VCF files

A major product of DNASeq experiments are catalogs of called variants (e.g., SNPs, indels). We will use the *VariantAnnotation* package to explore this type of data. Sample data included in the package are a subset of chromosome 22 from the 1000 Genomes project. Variant Call Format (VCF; full description) text files contain meta-information lines, a header line with column names, data lines with information about a position in the genome, and optional genotype information on samples for each position.

3.1 Coding consequences

Locating variants in and around genes Variant location with respect to genes can be identified with the `locateVariants` function. Regions are specified in the `region` argument and can be one of the following constructors: `CodingVariants()`, `IntronVariants()`, `FiveUTRVariants()`, `ThreeUTRVariants()`, `IntergenicVariants()`, `SpliceSiteVariants()`, or `AllVariants()`. Location definitions are shown in Table~3.

Exercise 6 *Load the TxDb.Hsapiens.UCSC.hg19.knownGene annotation package, and read in the `chr22.vcf.gz` example file from the VariantAnnotation package.*

Remembering to re-name sequence levels, use the `locateVariants` function to identify coding variants.

Summarize aspects of your data, e.g., did any coding variants match more than one gene? How many coding variants are there per gene ID?

Table 3: Variant locations

Location	Details
`coding`	Within a coding region
`fiveUTR`	Within a 5' untranslated region
`threeUTR`	Within a 3' untranslated region
`intron`	Within an intron region
`intergenic`	Not within a transcript associated with a gene
`spliceSite`	Overlaps any of the first or last 2 nucleotides of an intron

Solution: Here we open the known genes data base, and read in the VCF file.

```
library(VariantAnnotation)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")
vcf <- renameSeqlevels(vcf, c("22"="chr22"))
```

The next lines locate coding variants.

```
rd <- rowData(vcf)
loc <- locateVariants(rd, txdb, CodingVariants())
head(loc, 3)
```

```
## GRanges with 3 ranges and 7 metadata columns:
##       seqnames               ranges strand | LOCATION   QUERYID      TXID     CDSID
##          <Rle>            <IRanges>  <Rle> | <factor> <integer> <integer> <integer>
##   [1]    chr22 [50301422, 50301422]      * |   coding        24     75253    218562
##   [2]    chr22 [50301476, 50301476]      * |   coding        25     75253    218562
##   [3]    chr22 [50301488, 50301488]      * |   coding        26     75253    218562
##             GENEID       PRECEDEID        FOLLOWID
##       <character> <CharacterList> <CharacterList>
##   [1]       79087
##   [2]       79087
##   [3]       79087
##   ---
##   seqlengths:
##    chr22
##       NA
```

To answer gene-centric questions data can be summarized by gene regardless of transcript.

```
## Did any coding variants match more than one gene?
splt <- split(loc$GENEID, loc$QUERYID)
table(sapply(splt, function(x) length(unique(x)) > 1))
```

```
##
## FALSE  TRUE
##   965    15
```

```
## Summarize the number of coding variants by gene ID
splt <- split(loc$QUERYID, loc$GENEID)
head(sapply(splt, function(x) length(unique(x))), 3)
```

```
## 113730   1890  23209
##     22     15     30
```

Amino acid coding changes predictCoding computes amino acid coding changes for non-synonymous variants. Only ranges in query that overlap with a coding region in subject are considered. Reference sequences are retrieved from either a BSgenome or fasta file specified in seqSource. Variant sequences are constructed by substituting, inserting or deleting values in the varAllele column into the reference sequence. Amino acid codes are computed for the variant codon sequence when the length is a multiple of 3.

The query argument to predictCoding can be a GRanges or VCF. When a GRanges is supplied the varAllele argument must be specified. In the case of a VCF object, the alternate alleles are taken from alt(<VCF>) and the varAllele argument is not specified.

The result is a modified query containing only variants that fall within coding regions. Each row represents a variant-transcript match so more than one row per original variant is possible.

```
library(BSgenome.Hsapiens.UCSC.hg19)
coding <- predictCoding(vcf, txdb, seqSource=Hsapiens)
coding[5:9]

## GRanges with 5 ranges and 17 metadata columns:
##                     seqnames               ranges strand | paramRangeID            REF
##                        <Rle>            <IRanges>  <Rle> |     <factor> <DNAStringSet>
##   22:50301584_C/T    chr22 [50301584, 50301584]      - |         <NA>              C
##       rs114264124    chr22 [50302962, 50302962]      - |         <NA>              C
##       rs149209714    chr22 [50302995, 50302995]      - |         <NA>              C
##   22:50303554_T/C    chr22 [50303554, 50303554]      - |         <NA>              T
##        rs12167668    chr22 [50303561, 50303561]      - |         <NA>              C
##                                  ALT      QUAL      FILTER      varAllele     CDSLOC
##                     <DNAStringSetList> <numeric> <character> <DNAStringSet>  <IRanges>
##   22:50301584_C/T                  T       100        PASS              A [777, 777]
##       rs114264124                  T       100        PASS              A [698, 698]
##       rs149209714                  G       100        PASS              C [665, 665]
##   22:50303554_T/C                  C       100        PASS              G [652, 652]
##        rs12167668                  T       100        PASS              A [645, 645]
##                        PROTEINLOC   QUERYID        TXID     CDSID      GENEID   CONSEQUENCE
##                     <IntegerList> <integer> <character> <integer> <character>      <factor>
##   22:50301584_C/T           259        28       75253    218562       79087    synonymous
##       rs114264124           233        57       75253    218563       79087 nonsynonymous
##       rs149209714           222        58       75253    218563       79087 nonsynonymous
##   22:50303554_T/C           218        73       75253    218564       79087 nonsynonymous
##        rs12167668           215        74       75253    218564       79087    synonymous
##                           REFCODON       VARCODON         REFAA         VARAA
##                     <DNAStringSet> <DNAStringSet> <AAStringSet> <AAStringSet>
##   22:50301584_C/T            CCG            CCA             P             P
##       rs114264124            CGG            CAG             R             Q
##       rs149209714            GGA            GCA             G             A
##   22:50303554_T/C            ATC            GTC             I             V
##        rs12167668            CCG            CCA             P             P
##   ---
##   seqlengths:
##    chr22
##       NA
```

Using variant rs114264124 as an example, we see varAllele A has been substituted into the refCodon CGG to produce varCodon CAG. The refCodon is the sequence of codons necessary to make the variant allele substitution and therefore often includes more nucleotides than indicated in the range (i.e. the range is 50302962, 50302962, width of 1). Notice it is the second position in the refCodon that has been substituted. This position in the codon, the position of substitution, corresponds to genomic position 50302962. This genomic position maps to position

698 in coding region-based coordinates and to triplet 233 in the protein. This is a non-synonymous coding variant where the amino acid has changed from `R` (Arg) to `Q` (Gln).

When the resulting `varCodon` is not a multiple of 3 it cannot be translated. The consequence is considered a `frameshift` and `varAA` will be missing.

```
coding[coding$CONSEQUENCE == "frameshift"]

## GRanges with 2 ranges and 17 metadata columns:
##                       seqnames               ranges strand | paramRangeID            REF
##                          <Rle>            <IRanges>  <Rle> |     <factor> <DNAStringSet>
##   22:50317001_G/GCACT    chr22 [50317001, 50317001]      + |         <NA>              G
##   22:50317001_G/GCACT    chr22 [50317001, 50317001]      + |         <NA>              G
##                                      ALT      QUAL      FILTER      varAllele     CDSLOC
##                       <DNAStringSetList> <numeric> <character> <DNAStringSet>  <IRanges>
##   22:50317001_G/GCACT              GCACT       233        PASS          GCACT [808, 808]
##   22:50317001_G/GCACT              GCACT       233        PASS          GCACT [628, 628]
##                          PROTEINLOC   QUERYID        TXID     CDSID      GENEID
##                       <IntegerList> <integer> <character> <integer> <character>
##   22:50317001_G/GCACT           270       359       74357    216303       79174
##   22:50317001_G/GCACT           210       359       74358    216303       79174
##                       CONSEQUENCE       REFCODON       VARCODON         REFAA
##                          <factor> <DNAStringSet> <DNAStringSet> <AAStringSet>
##   22:50317001_G/GCACT  frameshift            GCC            GCC             A
##   22:50317001_G/GCACT  frameshift            GCC            GCC             A
##                               VARAA
##                       <AAStringSet>
##   22:50317001_G/GCACT
##   22:50317001_G/GCACT
##   ---
##   seqlengths:
##    chr22
##       NA
```

References

[1] Kathi Zarnack, Julian König, Mojca Tajnik, Iñigo Martincorena, Sebastian Eustermann, Isabelle Stévant, Alejandro Reyes, Simon Anders, Nicholas˜M Luscombe, and Jernej Ule. Direct competition between hnrnp c and u2af65 protects the transcriptome from the exonization of¡ i¿ alu¡/i¿ elements. *Cell*, 152(3):453–466, 2013. URL: http://europepmc.org/abstract/MED/23374342.

[2] H.˜Li and R.˜Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26:589–595, Mar 2010.

[3] H.˜Li and R.˜Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25:1754–1760, Jul 2009.

[4] B.˜Langmead, C.˜Trapnell, M.˜Pop, and S.˜L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, 10:R25, 2009.

[5] Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin˜T. Morgan, and Vincent˜J. Carey. Software for computing and annotating genomic ranges. *PLoS Comput Biol*, 9(8):e1003118, 08 2013. URL: http://dx.doi.org/10.1371%2Fjournal.pcbi.1003118, doi:10.1371/journal.pcbi.1003118.