GenomicRanges HOWTOs

Bioconductor Team

Edited: October 2013; Compiled: May 1, 2014

Contents

1 Introduction

This vignette is a collection of *HOWTOs*. Each *HOWTO* is a short section that demonstrates how to use the containers and operations implemented in the *GenomicRanges* and related packages (*IRanges*, *GenomicFeatures*, *Rsamtools*, and *Biostrings*) to perform a task typically found in the context of a high throughput sequence analysis.

The HOWTOs are self contained, independent of each other, and can be studied and reproduced in any order.

We assume the reader has some previous experience with *R* and with basic manipulation of `GRanges`, `GRangesList`, `Rle`, `RleList`, and `DataFrame` objects. See the "An Introduction to Genomic Ranges Classes" vignette located in the *GenomicRanges* package (in the same folder as this document) for an introduction to these containers.

Additional recommended readings after this vignette are the "Software for Computing and Annotating Genomic Ranges" paper[Lawrence et al. (2013)] and the "Counting reads with `summarizeOverlaps`" vignette located in the *GenomicAlignments* package.

To display the list of vignettes available in the *GenomicRanges*, use `browseVignettes("GenomicRanges")`.

2 How to read BAM files into R

As sample data we use the *pasillaBamSubset* data package which contains both a BAM file with single-end reads (untreated1_chr4) and a BAM file with paired-end reads (untreated3_chr4). Each file is a subset of chr4 from the "Pasilla" experiment. See ?*pasillaBamSubset* for details.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4() ## single-end reads
```

Several functions are available for reading BAM files into *R*:

```
  scanBam()
  readGAlignments()
  readGAlignmentPairs()
  readGAlignmentsList()
```

`scanBam` is a low-level function that returns a list of lists and is not discussed further here. For details see `?scanBam` in the *Rsamtools* package.

2.1 Single-end reads

Single-end reads can be loaded with the `readGAlignments` function.

```
> library(GenomicAlignments)
> gal <- readGAlignments(un1)
```

Data subsets can be specified by genomic position, field names, or flag criteria in the `ScanBamParam`. Here we input records that overlap position 1 to 5000 on the negative strand with `flag` and `cigar` as metadata columns.

```
> what <- c("flag", "cigar")
> which <- GRanges("chr4", IRanges(1, 5000))
> flag <- scanBamFlag(isMinusStrand = TRUE)
> param <- ScanBamParam(which=which, what=what, flag=flag)
> neg <- readGAlignments(un1, param=param)
> neg
```

```
GAlignments with 37 alignments and 2 metadata columns:
       seqnames strand       cigar    qwidth     start       end
          <Rle>  <Rle> <character> <integer> <integer> <integer>
   [1]     chr4      -         75M        75       892       966
   [2]     chr4      -         75M        75       919       993
   [3]     chr4      -         75M        75       967      1041
   ...      ...    ...         ...       ...       ...       ...
  [35]     chr4      -         75M        75      4997      5071
  [36]     chr4      -         75M        75      4998      5072
  [37]     chr4      -         75M        75      4999      5073
           width     njunc   |      flag       cigar
       <integer> <integer>   | <integer> <character>
   [1]        75         0   |        16         75M
   [2]        75         0   |        16         75M
   [3]        75         0    |        16         75M
   ...       ...       ... ...        ...         ...
  [35]        75         0    |        16         75M
  [36]        75         0    |        16         75M
  [37]        75         0    |        16         75M
  ---
  seqlengths:
      chr2L     chr2R     chr3L     chr3R      chr4      chrM      chrX   chrYHet
   23011544  21146708  24543557  27905053   1351857     19517  22422827    347038
```

Another approach to subsetting the data is to use `filterBam`. This function creates a new BAM file of records passing user-defined criteria. See `?filterBam` for details.

2.2 Paired-end reads

Paired-end reads can be loaded with `readGAlignmentPairs` or `readGAlignmentsList`. These functions use the same mate paring algorithm but output different objects.

Let's start with `readGAlignmentPairs`:

```
> un3 <- untreated3_chr4()
> gapairs <- readGAlignmentPairs(un3)
```

The `GAlignmentPairs` class holds only pairs; reads with no mate or with ambiguous pairing are discarded. Each list element holds exactly 2 records (a mated pair). Records can be accessed as the `first` and`last` segments in a template or as `left` and `right` alignments. See `?GAlignmentPairs` for details.

```
> gapairs
```

```
GAlignmentPairs with 75346 alignment pairs and 0 metadata columns:
        seqnames strand   :              ranges  --              ranges
           <Rle>  <Rle>   :           <IRanges>  --           <IRanges>
    [1]     chr4      +   :          [169, 205]  --        [ 326,  362]
    [2]     chr4      +   :          [943, 979]  --        [1086, 1122]
    [3]     chr4      +   :          [944, 980]  --        [1119, 1155]
    ...      ...    ... ...                 ... ...                 ...
[75344]     chr4      +   : [1348217, 1348253]  -- [1348215, 1348251]
[75345]     chr4      +   : [1349196, 1349232]  -- [1349326, 1349362]
[75346]     chr4      +   : [1349708, 1349744]  -- [1349838, 1349874]
---
seqlengths:
    chr2L     chr2R     chr3L     chr3R      chr4      chrM      chrX   chrYHet
 23011544  21146708  24543557  27905053   1351857     19517  22422827    347038
```

For `readGAlignmentsList`, mate pairing is performed when `asMates` is set to `TRUE` on the `BamFile` object, otherwise records are treated as single-end.

```
> galist <- readGAlignmentsList(BamFile(un3, asMates=TRUE))
```

`GAlignmentsList` is a more general 'list-like' structure that holds mate pairs as well as non-mates (i.e., singletons, records with unmapped mates etc.) A `mates` metadata column (accessed with `mcols`) indicates which records were paired and is set on both the individual `GAlignments` and the outer list elements.

```
> galist
```

```
GAlignmentsList of length 96632:
[[1]]
GAlignments with 2 alignments and 0 metadata columns:
      seqnames strand cigar qwidth start end width njunc
```

```
  [1]     chr4      +   37M     37   169  205    37     0
  [2]     chr4      -   37M     37   326  362    37     0

[[2]]
GAlignments with 2 alignments and 0 metadata columns:
      seqnames strand cigar qwidth start   end width njunc
  [1]     chr4      +   37M     37   946   982    37     0
  [2]     chr4      -   37M     37   986  1022    37     0

[[3]]
GAlignments with 2 alignments and 0 metadata columns:
      seqnames strand cigar qwidth start   end width njunc
  [1]     chr4      +   37M     37   943   979    37     0
  [2]     chr4      -   37M     37  1086  1122    37     0

...
<96629 more elements>
---
seqlengths:
    chr2L    chr2R    chr3L    chr3R     chr4     chrM     chrX  chrYHet
 23011544 21146708 24543557 27905053  1351857    19517 22422827   347038
```

Non-mated reads are returned as groups by QNAME and contain any number of records. Here the non-mate groups range in size from 1 to 9.

```
> non_mates <- galist[unlist(mcols(galist)$mates) == FALSE]
> table(elementLengths(non_mates))
```

```
< table of extent 0 >
```

2.3 Iterating with `yieldSize`

Large files can be iterated through in chunks by setting a `yieldSize` on the `BamFile`.

```
> bf <- BamFile(un1, yieldSize=100000)
```

Iteration through a BAM file requires that the file be opened, repeatedly queried inside a loop, then closed. Repeated calls to `readGAlignments` without opening the file first result in the same 100000 records returned each time.

```
> open(bf)
> cvg <- NULL
> repeat {
+     chunk <- readGAlignments(bf)
+     if (length(chunk) == 0L)
+         break
+     chunk_cvg <- coverage(chunk)
+     if (is.null(cvg)) {
+         cvg <- chunk_cvg
+     } else {
+         cvg <- cvg + chunk_cvg
+     }
+ }
> close(bf)
> cvg
```

```
RleList of length 8
$chr2L
```

```
integer-Rle of length 23011544 with 1 run
  Lengths: 23011544
  Values :        0

$chr2R
integer-Rle of length 21146708 with 1 run
  Lengths: 21146708
  Values :        0

$chr3L
integer-Rle of length 24543557 with 1 run
  Lengths: 24543557
  Values :        0

$chr3R
integer-Rle of length 27905053 with 1 run
  Lengths: 27905053
  Values :        0

$chr4
integer-Rle of length 1351857 with 122061 runs
  Lengths:  891   27    5   12   13   45 ...  106   75 1600   75 1659
  Values :    0    1    2    3    4    5 ...    0    1    0    1    0

...
<3 more elements>
```

3 How to prepare a table of read counts for RNA-Seq differential gene expression

Methods for RNA-Seq gene expression analysis generally require a table of counts that summarize the number of reads that overlap or 'hit' a particular gene. In this section we count with `summarizeOverlaps` and create a count table from the results.

Other packages that provide read counting are *Rsubread* and *easyRNASeq*. The *parathyroidSE* package vignette contains a workflow on counting and other common operations required for differential expression analysis.

3.1 Counting with `summarizeOverlaps`

As sample data we use *pasillaBamSubset* which contains both a single-end BAM (untreated1_chr4) and a paired-end BAM (untreated3_chr4). Each file is a subset of chr4 from the "Pasilla" experiment. See ?*pasillaBamSubset* for details.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4() ## single-end records
```

`summarizeOverlaps` requires the name of a BAM file(s) and an annotation to count against. The annotation must match the genome build the BAM records were aligned to. For the pasilla data this is dm3 Dmelanogaster which is available as a *Bioconductor* package. Load the package and extract the exon ranges by gene.

```
> library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
> exbygene <- exonsBy(TxDb.Dmelanogaster.UCSC.dm3.ensGene, "gene")
```

summarizeOverlaps automatically sets a yieldSize on large BAM files and iterates over them in chunks. When reading paired-end data set the singleEnd argument to FALSE. See ?summarizeOverlaps for details reguarding the count modes and additional arguments.

```
> library(GenomicAlignments)
> se <- summarizeOverlaps(exbygene, un1, mode="IntersectionNotEmpty")
```

The return object is a SummarizedExperiment with counts in the assays slot.

```
> class(se)
```

```
[1] "SummarizedExperiment"
attr(,"package")
[1] "GenomicRanges"
```

```
> head(table(assays(se)$counts))
```

```
    0     1     2     3     4     5 
15593     1     3     1     4     1 
```

The count vector is the same length as the annotation.

```
> identical(length(exbygene), length(assays(se)$counts))
```

```
[1] TRUE
```

The annotation is stored in the rowData slot.

```
> rowData(se)
```

```
GRangesList of length 15682:
$FBgn0000003 
GRanges with 1 range and 2 metadata columns:
      seqnames             ranges strand |   exon_id   exon_name
         <Rle>          <IRanges>  <Rle> | <integer> <character>
  [1]    chr3R [2648220, 2648518]      + |     45123        <NA>

$FBgn0000008 
GRanges with 13 ranges and 2 metadata columns:
       seqnames               ranges strand   | exon_id exon_name
   [1]    chr2R [18024494, 18024531]      +   |   20314      <NA>
   [2]    chr2R [18024496, 18024713]      +   |   20315      <NA>
   [3]    chr2R [18024938, 18025756]      +   |   20316      <NA>
   ...      ...                  ...    ... ...     ...       ...
  [11]    chr2R [18059821, 18059938]      +   |   20328      <NA>
  [12]    chr2R [18060002, 18060339]      +   |   20329      <NA>
  [13]    chr2R [18060002, 18060346]      +   |   20330      <NA>

...
<15680 more elements>
---
seqlengths:
     chr2L     chr2R     chr3L ...   chrXHet   chrYHet chrUextra
  23011544  21146708  24543557 ...    204112    347038  29004656
```

3.2 Retrieving annotations from AnnotationHub

When the annotation is not available as a GRanges or a *Bioconductor* package it may be available in AnnotationHub. Create a 'hub' and filter on Drosophila melanogaster.

```
> library(AnnotationHub)
> hub <- AnnotationHub()
> filters(hub) <- list(Species="Drosophila melanogaster")
```

There are 87 files that match Drosophila melanogaster.

```
> length(hub)
```

```
[1] 101
```

```
> head(names(hub))
```

```
[1] "ensembl.release.69.fasta.drosophila_melanogaster.cdna.Drosophila_melanogaster.BDGP5.69.cdna.all.fa.rz"
[2] "ensembl.release.69.fasta.drosophila_melanogaster.dna.Drosophila_melanogaster.BDGP5.69.dna.toplevel.fa
[3] "ensembl.release.69.fasta.drosophila_melanogaster.dna.Drosophila_melanogaster.BDGP5.69.dna_rm.toplevel
[4] "ensembl.release.69.fasta.drosophila_melanogaster.dna.Drosophila_melanogaster.BDGP5.69.dna_sm.toplevel
[5] "ensembl.release.69.fasta.drosophila_melanogaster.ncrna.Drosophila_melanogaster.BDGP5.69.ncrna.fa.rz"
[6] "ensembl.release.69.fasta.drosophila_melanogaster.pep.Drosophila_melanogaster.BDGP5.69.pep.all.fa.rz"
```

Retrieve a dm3 file as a GRanges.

```
> gr <- hub$goldenpath.dm3.database.ensGene_0.0.1.RData
> summary(gr)
```

```
 Length   Class    Mode 
  23017 GRanges      S4 
```

The metadata fields contain the details of file origin and content.

```
> names(metadata(gr)[[2]])
```

```
 [1] "BiocVersion"   "DataProvider"  "Description"   "Genome"       
 [5] "Tags"          "SourceUrl"     "SourceVersion" "Species"      
 [9] "RDataPath"     "RDataName"    
```

```
> metadata(gr)[[2]]$Tags
```

```
CharacterList of length 1
[["7161"]] ensGene UCSC track Gene Transcript Annotation
```

Split the GRanges by gene name to get a GRangesList of transcripts by gene.

```
> split(gr, gr$name)
```

```
GRangesList of length 23017:
$FBtr0005009 
GRanges with 1 range and 5 metadata columns:
      seqnames             ranges strand |        name     score
         <Rle>          <IRanges>  <Rle> | <character> <numeric>
  [1]    chr2R [9134178, 9135136]      + | FBtr0005009         0
          itemRgb              thick                          blocks
      <character>          <IRanges>                   <IRangesList>
  [1]        <NA> [9134248, 9135013] [  1, 100] [245, 577] [645, 959]

$FBtr0005088 
GRanges with 1 range and 5 metadata columns:
      seqnames             ranges strand |        name score itemRgb
  [1]    chr2L [8366009, 8370085]      + | FBtr0005088     0    <NA>
                   thick                                   blocks
  [1] [8366311, 8369720] [   1,  386] [1088, 1241] [1304, 1722] ...

$FBtr0005673 
```

```
GRanges with 1 range and 5 metadata columns:
      seqnames             ranges strand |        name score itemRgb
  [1]    chr2L [8438269, 8442352]      + | FBtr0005673     0    <NA>
                   thick                                      blocks
  [1] [8438376, 8442310] [   1,  434] [ 504, 2663] [2756, 4084]

...
<23014 more elements>
---
seqlengths:
     chr2L   chr2LHet     chr2R ...    chrXHet    chrYHet       chrM
  23011544     368872  21146708 ...     204112     347038      19517
```

Before performing overlap operations confirm that the seqlevels (chromosome names) in the annotation match those in the BAM file. See `?renameSeqlevels`, `?keepSeqlevels` and `?seqlevels` for examples of renaming seqlevels.

3.3 Count tables

Two popular packages for gene expression are *DESeq* and *edgeR*. Tables of counts per gene are required for both and can be easily created with a vector of counts. Here we use the counts from the `SummarizedExperiment`.

```
> library(DESeq)
> deseq <- newCountDataSet(assays(se)$counts, rownames(colData(se)))
> library(edgeR)
> edger <- DGEList(assays(se)$counts, group=rownames(colData(se)))
```

4 How to extract DNA sequences of gene regions

4.1 DNA sequences for intron and exon regions of a single gene

DNA sequences for the introns and exons of a gene are essentially the sequences for the introns and exons for all known transcripts of a gene. The first task is to identify all transcripts associated with the gene of interest. Our sample gene is the human TRAK2 which is involved in regulation of endosome-to-lysosome trafficking of membrane cargo. The Entrez gene id is '66008'.

```
> trak2 <- "66008"
```

Load the UCSC 'Known Gene' table annotation available as a *Bioconductor* package.

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

To get the transcripts associated with the trak2 gene we use the `transcriptsBy` function from the *GenomicFeatures* package. This returns a `GRangesList` of all transcripts grouped by gene. We are only interested in trak2 so we subset the list on the trak2 gene id.

```
> library(GenomicFeatures)
> txbygene <- transcriptsBy(txdb, by="gene")[trak2]
> txbygene
```

```
GRangesList of length 1:
$66008
GRanges with 2 ranges and 2 metadata columns:
      seqnames                 ranges strand |     tx_id     tx_name
         <Rle>              <IRanges>  <Rle> | <integer> <character>
```

```
  [1]     chr2 [202241930, 202316319]      - |     12552  uc002uyb.4
  [2]     chr2 [202259851, 202316319]      - |     12553  uc002uyc.2

---
seqlengths:
                 chr1                 chr2 ...       chrUn_gl000249
            249250621            243199373 ...                38502
```

The transcript names corresponding to the trak2 gene will be used to subset the extracted intron and exon regions. The `txbygene` object is a `GRangesList` and the transcript names are a metadata column on the individual `GRanges`. To extract the names we must first 'flatten' or unlist `txbygene`.

```
> tx_names <- mcols(unlist(txbygene))$tx_name
> tx_names
```

```
[1] "uc002uyb.4" "uc002uyc.2"
```

Intron and exon regions are extracted with `intronsByTranscript` and `exonsBy`. The resulting `GRangesLists` are subset on the trak2 transcript names.

Extract the intron regions ...

```
> intronsbytx <- intronsByTranscript(txdb, use.names=TRUE)[tx_names]
> elementLengths(intronsbytx)
```

```
uc002uyb.4 uc002uyc.2 
        15          7 
```

and the exon regions.

```
> exonsbytx <- exonsBy(txdb, "tx", use.names=TRUE)[tx_names]
> elementLengths(exonsbytx)
```

```
uc002uyb.4 uc002uyc.2 
        16          8 
```

Next we want the DNA sequences for these intron and exon regions. The `extractTranscriptSeqs` function in the *Biostrings* package will query a *BSGenome* package with a set of genomic positions and retrieve the DNA sequences.

```
> library(Biostrings)
> library(BSgenome.Hsapiens.UCSC.hg19)
```

Extract the intron sequences ...

```
> intron_seqs <- extractTranscriptSeqs(Hsapiens, intronsbytx)
> intron_seqs
```

```
  A DNAStringSet instance of length 2
    width seq                                        names               
[1] 67863 GTAAGAGTGCCTGGGAAAT...CTTGATGTTTTTGTTTTAG uc002uyb.4
[2] 54937 GTGAGTATTAACATATTCT...CTTGATGTTTTTGTTTTAG uc002uyc.2
```

and the exon sequences.

```
> exon_seqs <- extractTranscriptSeqs(Hsapiens, exonsbytx)
> exon_seqs
```

```
  A DNAStringSet instance of length 2
    width seq                                        names               
[1]  6527 GCTGGGAGAGTGGCTCTCC...TGAGTAGCTTGAATTTTCA uc002uyb.4
[2]  1532 GCTGGGAGAGTGGCTCTCC...AATAAATACTTTCAAGTCA uc002uyc.2
```

4.2 DNA sequences for coding and UTR regions of genes associated with colorectal cancer

In this section we extract the coding and UTR sequences of genes involved in colorectal cancer. The workflow extends the ideas presented in the single gene example and suggests an approach to identify disease-related genes.

4.2.1 Build a gene list

We start with a list of gene or transcript ids. If you do not have pre-defined list one can be created with the *KEGG.db* and *KEGGgraph* packages. Updates to the data in the *KEGG.db* package are no longer available, however, the resource is still useful for identifying pathway names and ids.

Create a table of KEGG pathways and ids and search on the term 'cancer'.

```
> library(KEGG.db)
> pathways <- toTable(KEGGPATHNAME2ID)
> pathways[grepl("cancer", pathways$path_name, fixed=TRUE),]
```

```
    path_id                   path_name
299   05200          Pathways in cancer
300   05210           Colorectal cancer
302   05212           Pancreatic cancer
303   05213          Endometrial cancer
305   05215             Prostate cancer
306   05216              Thyroid cancer
309   05219              Bladder cancer
312   05222      Small cell lung cancer
313   05223  Non-small cell lung cancer
```

Use the "05210" id to query the KEGG web resource (accesses the currently maintained data).

```
> library(KEGGgraph)
> dest <- tempfile()
> retrieveKGML("05200", "hsa", dest, "internal")
```

The suffix of the KEGG id is the Entrez gene id. The `translateKEGGID2GeneID` simply removes the prefix leaving just the Entrez gene ids.

```
> crids <- as.character(parseKGML2DataFrame(dest)[,1])
> crgenes <- unique(translateKEGGID2GeneID(crids))
> head(crgenes)
```

```
[1] "1630"  "836"   "842"   "1499"  "51384" "54361"
```

4.2.2 Identify genomic coordinates

The list of gene ids is used to extract genomic positions of the regions of interest. The Known Gene table from UCSC will be the annotation and is available as a *Bioconductor* package.

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

If an annotation is not available as a *Bioconductor* annotation package it may be available in *AnnotationHub*. Additionally, there are functions in *GenomicFeatures* which can retrieve data from UCSC and Ensembl to create a `TranscriptDb`. See ?`makeTranscriptDbFromUCSC` for details.

As in the single gene example we need to identify the transcripts corresponding to each gene. The transcript id (or name) is used to isolate the UTR and coding regions of interest. This grouping of transcript by gene is also used to re-group the final sequence results.

The transcriptsBy function outputs both the gene and transcript identifiers which we use to create a map between the two. The map is a CharacterList with gene ids as names and transcript ids as the list elements.

```
> txbygene <- transcriptsBy(txdb, "gene")[crgenes] ## subset on colorectal genes
> map <- relist(unlist(txbygene, use.names=FALSE)$tx_id, txbygene)
> map
```

```
IntegerList of length 239
[["1630"]] 64962 64963 64964
[["836"]] 20202 20203 20204
[["842"]] 4447 4448 4449 4450 4451 4452
[["1499"]] 13582 13583 13584 13585 13586 13587 13589
[["51384"]] 29319 29320 29321
[["54361"]] 4634 4635
[["7471"]] 46151
[["7472"]] 31279 31280
[["7473"]] 63770
[["7474"]] 16089 16090 16091 16092
...
<229 more elements>
```

Extract the UTR and coding regions.

```
> cds <- cdsBy(txdb, "tx")
> threeUTR <- threeUTRsByTranscript(txdb)
> fiveUTR <- fiveUTRsByTranscript(txdb)
```

Coding and UTR regions may not be present for all transcripts specified in map. Consequently, the subset results will not be the same length. This length discrepancy must be taken into account when re-listing the final results by gene.

```
> txid <- unlist(map, use.names=FALSE)
> cds <- cds[names(cds) %in% txid]
> threeUTR <- threeUTR[names(threeUTR) %in% txid]
> fiveUTR <- fiveUTR[names(fiveUTR) %in% txid]
```

Note the different lengths of the subset regions.

```
> length(txid) ## all possible transcripts

[1] 1045

> length(cds)

[1] 960

> length(threeUTR)

[1] 919

> length(fiveUTR)

[1] 947
```

These objects are GRangesLists with the transcript id as the outer list element.

```
> cds

GRangesList of length 960:
$2043
GRanges with 6 ranges and 3 metadata columns:
      seqnames                 ranges strand |    cds_id    cds_name
         <Rle>              <IRanges>  <Rle> | <integer> <character>
  [1]     chr1 [113010160, 113010213]      + |      6055        <NA>
  [2]     chr1 [113033633, 113033703]      + |      6056        <NA>
  [3]     chr1 [113057496, 113057716]      + |      6058        <NA>
  [4]     chr1 [113058762, 113059039]      + |      6060        <NA>
  [5]     chr1 [113059743, 113060007]      + |      6061        <NA>
  [6]     chr1 [113062902, 113063131]      + |      6062        <NA>
      exon_rank
      <integer>
  [1]         1
  [2]         2
  [3]         3
  [4]         4
  [5]         5
  [6]         6

$2044
GRanges with 4 ranges and 3 metadata columns:
      seqnames                 ranges strand | cds_id cds_name
  [1]     chr1 [113057590, 113057716]      + |   6059     <NA>
  [2]     chr1 [113058762, 113059039]      + |   6060     <NA>
  [3]     chr1 [113059743, 113060007]      + |   6061     <NA>
  [4]     chr1 [113062902, 113063131]      + |   6062     <NA>
      exon_rank
  [1]         2
  [2]         3
  [3]         4
  [4]         5

$2045
GRanges with 5 ranges and 3 metadata columns:
      seqnames                 ranges strand | cds_id cds_name
  [1]     chr1 [113051885, 113052066]      + |   6057     <NA>
  [2]     chr1 [113057496, 113057716]      + |   6058     <NA>
  [3]     chr1 [113058762, 113059039]      + |   6060     <NA>
  [4]     chr1 [113059743, 113060007]      + |   6061     <NA>
  [5]     chr1 [113062902, 113063131]      + |   6062     <NA>
      exon_rank
  [1]         1
  [2]         2
  [3]         3
  [4]         4
  [5]         5

...
<957 more elements>
---
seqlengths:
                 chr1                 chr2 ...       chrUn_gl000249
            249250621            243199373 ...                38502
```

4.2.3 Extract sequences from BSgenome

The BSgenome packages contain complete genome sequences for a given organism.

Load the BSgenome package for homo sapiens.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> genome <- BSgenome.Hsapiens.UCSC.hg19
```

Use `extractTranscriptSeqs` to extract the UTR and coding regions from the `BSGenome`. This function retrieves the sequences for an any `GRanges` or `GRangesList` (i.e., not just transcripts like the name implies).

```
> threeUTR_seqs <- extractTranscriptSeqs(genome, threeUTR)
> fiveUTR_seqs <- extractTranscriptSeqs(genome, fiveUTR)
> cds_seqs <- extractTranscriptSeqs(genome, cds)
```

The return values are `DNAStringSet` objects.

```
> cds_seqs

  A DNAStringSet instance of length 960
      width seq                                      names
  [1]  1119 ATGTTGGATGGCCTTGGA...TGGCTGGACCAAACCTGA 2043
  [2]   900 ATGCGTTCAGTGGGCGAG...TGGCTGGACCAAACCTGA 2044
  [3]  1176 ATGCTGAGACCGGGTGGT...TGGCTGGACCAAACCTGA 2045
  ...   ... ...
[958]   681 ATGTTACGACAAGATTCC...CACAATGAATCAACGTAG 78103
[959]   768 ATGAGTGGAAAGGTGACC...CACAATGAATCAACGTAG 78104
[960]   600 ATGAGTGGAAAGGTGACC...CACAATGAATCAACGTAG 78105
```

Our final step is to collect the coding and UTR regions (currently organzied by transcript) into groups by gene id. The `relist` function groups the sequences of a `DNAStringSet` object into a `DNAStringSetList` object, based on the specified `skeleton` argument. The `skeleton` must be a list-like object and only its shape (i.e. its element lengths) matters (its exact content is ignored). A simple form of `skeleton` is to use a partitioning object that we make by specifying the size of each partition. The partitioning objects are different for each type of region because not all transcripts had a coding or 3' or 5' UTR region defined.

```
> lst3 <- relist(threeUTR_seqs, PartitioningByWidth(sum(map %in% names(threeUTR))))
> lst5 <- relist(fiveUTR_seqs, PartitioningByWidth(sum(map %in% names(fiveUTR))))
> lstc <- relist(cds_seqs, PartitioningByWidth(sum(map %in% names(cds))))
```

There are 239 genes in `map` each of which have 1 or more transcripts. The table of element lengths shows how many genes have each number of transcripts. For example, 47 genes have 1 transcript, 48 genes have 2 etc.

```
> length(map)

[1] 239

> table(elementLengths(map))

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 21 30
47 48 46 22 17 18 10  4  3  3  5  3  1  1  1  1  4  1  2  1  1
```

The lists of DNA sequences all have the same length as `map` but one or more of the element lengths may be zero. This would indicate that data were not available for that gene. The tables below show that there was at least 1 coding region available for all genes (i.e., none of the element lengths are 0). However, both the 3' and 5' UTR results have element lengths of 0 which indicates no UTR data were available for that gene.

```
> table(elementLengths(lstc))

 1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 30
48 54 49 20 17 16  8  5  5  3  1  2  3  1  2  1  3  1

> table(elementLengths(lst3))

 0  1  2  3  4  5  6  7  8  9 11 12 13 14 15 16 17 18 30
 2 49 56 47 19 18 13  9  5  8  2  2  2  1  1  2  1  1  1

> names(lst3)[elementLengths(lst3) == 0L] ## genes with no 3' UTR data
```

```
[1] "2255" "8823"
```

```
> table(elementLengths(lst5))
```

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 30
 3 48 52 49 19 17 16  8  5  5  3  2  2  3  1  1  1  3  1
```

```
> names(lst5)[elementLengths(lst5) == 0L] ## genes with no 5' UTR data
```

```
[1] "2255"  "27006" "8823"
```

5 How to create DNA consensus sequences for read group 'families'

The motivation for this HOWTO comes from a study which explored the dynamics of point mutations. The mutations of interest exist with a range of frequencies in the control group (e.g., 0.1% - 50%). PCR and sequencing error rates make it difficult to identify low frequency events (e.g., $< 20\%$).

When a library is prepared with Nextera, random fragments are generated followed by a few rounds of PCR. When the genome is large enough, reads aligning to the same start position are likely descendant from the same template fragment and should have identical sequences.

The goal is to elimininate noise by grouping the reads by common start position and discarding those that do not exceed a certain threshold within each family. A new consensus sequence will be created for each read group family.

5.1 Sort reads into groups by start position

Load the BAM file into a GAlignments object.

```
> library(Rsamtools)
> bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools")
> param <- ScanBamParam(what=c("seq", "qual"))
> gal <- readGAlignmentsFromBam(bamfile, use.names=TRUE, param=param)
```

Use the `sequenceLayer` function to *lay* the query sequences and quality strings on the reference.

```
> qseq <- setNames(mcols(gal)$seq, names(gal))
> qual <- setNames(mcols(gal)$qual, names(gal))
> qseq_on_ref <- sequenceLayer(qseq, cigar(gal),
+                              from="query", to="reference")
> qual_on_ref <- sequenceLayer(qual, cigar(gal),
+                              from="query", to="reference")
```

Split by chromosome.

```
> qseq_on_ref_by_chrom <- splitAsList(qseq_on_ref, seqnames(gal))
> qual_on_ref_by_chrom <- splitAsList(qual_on_ref, seqnames(gal))
> pos_by_chrom <- splitAsList(start(gal), seqnames(gal))
```

For each chromosome generate one GRanges object that contains unique alignment start positions and attach 3 metadata columns to it: the number of reads, the query sequences, and the quality strings.

```
> gr_by_chrom <- lapply(seqlevels(gal),
+   function(seqname)
+   {
+     qseq_on_ref2 <- qseq_on_ref_by_chrom[[seqname]]
+     qual_on_ref2 <- qual_on_ref_by_chrom[[seqname]]
+     pos2 <- pos_by_chrom[[seqname]]
+     qseq_on_ref_per_pos <- split(qseq_on_ref2, pos2)
+     qual_on_ref_per_pos <- split(qual_on_ref2, pos2)
+     nread <- elementLengths(qseq_on_ref_per_pos)
+     gr_mcols <- DataFrame(nread=unname(nread),
+                           qseq_on_ref=unname(qseq_on_ref_per_pos),
+                           qual_on_ref=unname(qual_on_ref_per_pos))
+     gr <- GRanges(Rle(seqname, nrow(gr_mcols)),
+                   IRanges(as.integer(names(nread)), width=1))
+     mcols(gr) <- gr_mcols
+     seqlevels(gr) <- seqlevels(gal)
+     gr
+   })
```

Combine all the GRanges objects obtained in (4) in 1 big GRanges object:

```
> gr <- do.call(c, gr_by_chrom)
> seqinfo(gr) <- seqinfo(gal)
```

'gr' is a GRanges object that contains unique alignment start positions:

```
> gr[1:6]
```

```
GRanges with 6 ranges and 3 metadata columns:
      seqnames    ranges strand |     nread
         <Rle> <IRanges>  <Rle> | <integer>
  [1]     seq1  [ 1,  1]      * |         1
  [2]     seq1  [ 3,  3]      * |         1
  [3]     seq1  [ 5,  5]      * |         1
  [4]     seq1  [ 6,  6]      * |         1
  [5]     seq1  [ 9,  9]      * |         1
  [6]     seq1  [13, 13]      * |         2
                                                                    qseq_on_ref
                                                             <DNAStringSetList>
  [1]                                      CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG
  [2]                                       CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT
  [3]                                       AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC
  [4]                                      GTGGCTCATTGTAATTTTTTGTTTTAACTCTTCTCT
  [5]                                       GCTCATTGTAAATGTGTGGTTTAACTCGTCCATGG
  [6] ATTGTAAATGTGTGGTTTAACTCGTCCCTGGCCCA,ATTGTAAATGTGTGGTTTAACTCGTCCATGGCCCAG
                                                                    qual_on_ref
                                                               <BStringSetList>
  [1]                                      <<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<;:<;7
  [2]                                       <<<<<<<<<<0<<<<655<<7<<<:9<<3/:<6):
  [3]                                       <<<<<<<<<<<7;71<<;<;;<7;<<3;);3*8/5
  [4]                                      (-&----,----)-)-),'--)---',+-,),''*,
  [5]                                       <<<<<<<<<<<<<<<;<;7<<<<<<<<7<<;:<5%
  [6] <<<<<<<<<;<<<8<<<<<;8:;6/686&;(16666,<<<<<;<<<;<;<<<<<<<<<<<8<8<3<8;<;<0;
  ---
  seqlengths:
   seq1 seq2
   1575 1584
```

Look at qseq_on_ref and qual_on_ref.

```
> qseq_on_ref
```

```
  A DNAStringSet instance of length 3271
       width seq                                     names
   [1]    36 CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG   B7_591:4:96:693:509
   [2]     35 CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT    EAS54_65:7:152:36...
   [3]     35 AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC    EAS51_64:8:5:734:57
   ...    ... ...
[3269]     35 TTTTTTCTTTTTTTTTTTTTTTTTTTTGCATGCCA    EAS139_11:7:50:12...
[3270]     35 TTTTTTTTTTTTTTTTTTTTTTTTTGCATGCCAGAAA    EAS54_65:3:320:20...
[3271]     35 TTTTTTTTTTTTTTTTTTTTTTTTTCATGCCAGAAAA    EAS114_26:7:37:79...
```

```
> qual_on_ref
```

```
  A BStringSet instance of length 3271
       width seq                                  names
   [1]     36 <<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<;:<;7 B7_591:4:96:693:509
   [2]     35 <<<<<<<<<<0<<<<655<<7<<<:9<<3/:<6):  EAS54_65:7:152:36...
   [3]     35 <<<<<<<<<<<7;71<<;<;;<7;<<3;);3*8/5  EAS51_64:8:5:734:57
   ...    ... ...
[3269]     35 <<<<,<&<7<<<<<<<<<<<<<<<<<<<<<<<<<<  EAS139_11:7:50:12...
[3270]     35 +'''/<<<<7:;+<;::<<<;;<<<<<<<<<<<<<  EAS54_65:3:320:20...
[3271]     35 3,,,===6===<===<;=====-============  EAS114_26:7:37:79...
```

2 reads align to start position 13. Let's have a close look at their sequences:

```
> mcols(gr)$qseq_on_ref[[6]]
```

```
  A DNAStringSet instance of length 2
    width seq                                     names
[1]    35 ATTGTAAATGTGTGGTTTAACTCGTCCCTGGCCCA     EAS56_61:6:18:467...
[2]    36 ATTGTAAATGTGTGGTTTAACTCGTCCATGGCCCAG    EAS114_28:5:296:3...
```

and their qualities:

```
> mcols(gr)$qual_on_ref[[6]]
```

```
  A BStringSet instance of length 2
    width seq                                     names
[1]    35 <<<<<<<<;<<<8<<<<<;8:;6/686&;(16666     EAS56_61:6:18:467...
[2]    36 <<<<<;<<<;<;<<<<<<<<<<<8<8<3<8;<;<0;    EAS114_28:5:296:3...
```

Note that the sequence and quality strings are those projected to the reference so the first letter in those strings are on top of start position 13, the 2nd letter on top of position 14, etc...

5.2 Remove low frequency reads

For each start position, remove reads with and under-represented sequence (e.g. threshold = 20% for the data used here which is low coverage). A unique number is assigned to each unique sequence. This will make future calculations easier and a little bit faster.

```
> qseq_on_ref <- mcols(gr)$qseq_on_ref
> tmp <- unlist(qseq_on_ref, use.names=FALSE)
> qseq_on_ref_id <- relist(match(tmp, tmp), qseq_on_ref)
```

Quick look at 'qseq_on_ref_id': It's an IntegerList object with the same length and "shape" as 'qseq_on_ref'.

```
> qseq_on_ref_id
```

```
IntegerList of length 1934
[[1]] 1
[[2]] 2
[[3]] 3
[[4]] 4
[[5]] 5
```

```
[[6]] 6 7
[[7]] 8
[[8]] 9
[[9]] 10 11
[[10]] 12
...
<1924 more elements>
```

Remove the under represented ids from each list element of 'qseq_on_ref_id':

```
> qseq_on_ref_id2 <- endoapply(qseq_on_ref_id,
+     function(ids) ids[countMatches(ids, ids) >= 0.2 * length(ids)])
```

Remove corresponding sequences from 'qseq_on_ref':

```
> tmp <- unlist(qseq_on_ref_id2, use.names=FALSE)
> qseq_on_ref2 <- relist(unlist(qseq_on_ref, use.names=FALSE)[tmp],
+                        qseq_on_ref_id2)
```

5.3 Create a consensus sequence for each read group family

Compute 1 consensus matrix per chromosome:

```
> split_factor <- rep.int(seqnames(gr), elementLengths(qseq_on_ref2))
> qseq_on_ref2 <- unlist(qseq_on_ref2, use.names=FALSE)
> qseq_on_ref2_by_chrom <- splitAsList(qseq_on_ref2, split_factor)
> qseq_pos_by_chrom <- splitAsList(start(gr), split_factor)
> cm_by_chrom <- lapply(names(qseq_pos_by_chrom),
+     function(seqname)
+         consensusMatrix(qseq_on_ref2_by_chrom[[seqname]],
+                         as.prob=TRUE,
+                         shift=qseq_pos_by_chrom[[seqname]]-1,
+                         width=seqlengths(gr)[[seqname]]))
> names(cm_by_chrom) <- names(qseq_pos_by_chrom)
```

'cm_by_chrom' is a list of consensus matrices. Each matrix has 17 rows (1 per letter in the DNA alphabet) and 1 column per chromosome position.

```
> lapply(cm_by_chrom, dim)
```

```
$seq1
[1]   18 1575

$seq2
[1]   18 1584
```

Compute the consensus string from each consensus matrix. We'll put "+" in the strings wherever there is no coverage for that position, and "N" where there is coverage but no consensus.

```
> cs_by_chrom <- lapply(cm_by_chrom,
+     function(cm) {
+         ## need to "fix" 'cm' because consensusString()
+         ## doesn't like consensus matrices with columns
+         ## that contain only zeroes (e.g., chromosome
+         ## positions with no coverage)
+         idx <- colSums(cm) == 0L
+         cm["+", idx] <- 1
+         DNAString(consensusString(cm, ambiguityMap="N"))
+     })
```

The new consensus strings.

```
> cs_by_chrom

$seq1
  1575-letter "DNAString" instance
seq: NANTAGNNNCTCANTTTAAANNTTTNTTTTTN...AATNATANNTTTNTTNTTTNTCTGNAC+++++

$seq2
  1584-letter "DNAString" instance
seq: ++++++++++++++++++++++++++++++++...NNNANANANANCTNNA++++++++++++++++
```

6 How to compute binned averages along a genome

In some applications, there is the need to compute the average of a variable along a genome for a set of predefined fixed-width regions (sometimes called "bins"). One such example is coverage. Coverage is an `RleList` with one list element per chromosome. Here we simulate a coverage list.

```
> library(BSgenome.Scerevisiae.UCSC.sacCer2)
> set.seed(22)
> cov <- RleList(
+     lapply(seqlengths(Scerevisiae),
+            function(len) Rle(sample(-10:10, len, replace=TRUE))),
+     compress=FALSE)
> head(cov, 3)

RleList of length 3
$chrI
integer-Rle of length 230208 with 219146 runs
  Lengths:   1   1   1   1   1   1   1 ...   1   1   1   1   1   1   1
  Values :  -4  -1  10   0   7   5   2 ...   4  -2  -8   1 -10  -8 -10

$chrII
integer-Rle of length 813178 with 774522 runs
  Lengths:   1   1   1   1   1   1   1 ...   1   1   1   2   2   1   1
  Values :  -3  -6  -7  -3   9  -4 -10 ...  -3  -4  -5   2  -2  -8   0

$chrIII
integer-Rle of length 316617 with 301744 runs
  Lengths:   1   1   1   1   1   1   1 ...   1   1   1   1   1   1   1
  Values :   2  -3  -6   5   9   5   3 ...   4  -7 -10  -5 -10  -1  -3
```

Use the `tileGenome` function to create a set of bins along the genome.

```
> bins1 <- tileGenome(seqinfo(Scerevisiae), tilewidth=100,
+                     cut.last.tile.in.chrom=TRUE)
```

We define the following function to compute the binned average of a numerical variable defined along a genome.

```
Arguments:
  'bins': a GRanges object representing the genomic bins.
       Typically obtained by calling tileGenome() with
       'cut.last.tile.in.chrom=TRUE'.
  'numvar': a named RleList object representing a numerical
```

```
       variable defined along the genome covered by 'bins', which
       is the genome described by 'seqinfo(bins)'.
  'mcolname': the name to give to the metadata column that will
       contain the binned average in the returned object.
```

The function returns 'bins' with an additional metadata column named 'mcolname' containing the binned average.

```
> binnedAverage <- function(bins, numvar, mcolname)
+ {
+     stopifnot(is(bins, "GRanges"))
+     stopifnot(is(numvar, "RleList"))
+     stopifnot(identical(seqlevels(bins), names(numvar)))
+     bins_per_chrom <- split(ranges(bins), seqnames(bins))
+     means_list <- lapply(names(numvar),
+         function(seqname) {
+             views <- Views(numvar[[seqname]],
+                            bins_per_chrom[[seqname]])
+             viewMeans(views)
+         })
+     new_mcol <- unsplit(means_list, as.factor(seqnames(bins)))
+     mcols(bins)[[mcolname]] <- new_mcol
+     bins
+ }
```

Compute the binned average for 'cov':

```
> bins1 <- binnedAverage(bins1, cov, "binned_cov")
> bins1
```

```
GRanges with 121639 ranges and 1 metadata column:
           seqnames       ranges strand   |         binned_cov
              <Rle>    <IRanges>  <Rle>   |          <numeric>
       [1]     chrI   [  1, 100]      *   |              -0.66
       [2]     chrI   [101, 200]      *   |              -0.05
       [3]     chrI   [201, 300]      *   |              -1.56
       ...      ...          ...    ... ...                ...
  [121637]  2micron [6101, 6200]      *   |              -0.25
  [121638]  2micron [6201, 6300]      *   |              -0.54
  [121639]  2micron [6301, 6318]      *   | -0.444444444444444
  ---
  seqlengths:
      chrI    chrII  chrIII   chrIV ...   chrXV  chrXVI    chrM 2micron
    230208   813178  316617 1531919 ... 1091289  948062   85779    6318
```

The bin size can be modified with the `tilewidth` argument to `tileGenome`. For additional examples see `?tileGenome`.

7 Session Information

```
R version 3.1.0 (2014-04-10)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
 [1] BSgenome.Scerevisiae.UCSC.sacCer2_1.3.99
 [2] KEGGgraph_1.22.1
 [3] graph_1.42.0
 [4] XML_3.98-1.1
 [5] KEGG.db_2.14.0
 [6] RSQLite_0.11.4
 [7] DBI_0.2-7
 [8] BSgenome.Hsapiens.UCSC.hg19_1.3.99
 [9] TxDb.Hsapiens.UCSC.hg19.knownGene_2.14.0
[10] edgeR_3.6.1
[11] limma_3.20.1
[12] DESeq_1.16.0
[13] lattice_0.20-29
[14] locfit_1.5-9.1
[15] AnnotationHub_1.4.0
[16] TxDb.Dmelanogaster.UCSC.dm3.ensGene_2.14.0
[17] GenomicFeatures_1.16.0
[18] AnnotationDbi_1.26.0
[19] Biobase_2.24.0
[20] GenomicAlignments_1.0.1
[21] BSgenome_1.32.0
[22] Rsamtools_1.16.0
[23] Biostrings_2.32.0
[24] XVector_0.4.0
[25] GenomicRanges_1.16.3
[26] GenomeInfoDb_1.0.2
[27] IRanges_1.22.5
[28] BiocGenerics_0.10.0
[29] pasillaBamSubset_0.2.0

loaded via a namespace (and not attached):
 [1] BBmisc_1.6               BatchJobs_1.2
 [3] BiocInstaller_1.14.2     BiocParallel_0.6.0
 [5] BiocStyle_1.2.0          Category_2.30.0
 [7] GSEABase_1.26.0          MASS_7.3-32
 [9] Matrix_1.1-3             RBGL_1.40.0
[11] RColorBrewer_1.0-5       RCurl_1.95-4.1
[13] RJSONIO_1.2-0.2          Rcpp_0.11.1
[15] annotate_1.42.0          biomaRt_2.20.0
[17] bitops_1.0-6             brew_1.0-6
[19] caTools_1.17             codetools_0.2-8
[21] colorspace_1.2-4         digest_0.6.4
[23] fail_1.2                 foreach_1.4.2
[25] genefilter_1.46.0        geneplotter_1.42.0
[27] ggplot2_0.9.3.1          grid_3.1.0
```

```
[29] gridSVG_1.4-0            gtable_0.1.2
[31] httpuv_1.3.0             httr_0.3
[33] interactiveDisplay_1.2.0 iterators_1.0.7
[35] munsell_0.4.2            plyr_1.8.1
[37] proto_0.3-10             reshape2_1.4
[39] rjson_0.2.13             rtracklayer_1.24.0
[41] scales_0.2.4             sendmailR_1.1-2
[43] shiny_0.9.1              splines_3.1.0
[45] stats4_3.1.0             stringr_0.6.2
[47] survival_2.37-7          tools_3.1.0
[49] xtable_1.7-3             zlibbioc_1.10.0
```

References

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 Computational Biology*, 4(3), 2013.