Practical: Ranges

Martin Morgan (mtmorgan@fhcrc.org)

27-28 February, 2014

Contents

1 Working with ranges 1
  1.1 Selecting gene sequences .......................................................... 6
  1.2 Summarizing overlaps ................................................................. 7

1 Working with ranges

Start by loading the GenomicRanges package and defining the plotRanges helper function.

Ranges describe both features of interest (e.g., genes, exons, promoters) and reads aligned to the genome. Bioconductor has very powerful facilities for working with ranges, some of which are summarized in Table 1. These are implemented in the GenomicRanges package; see [1] for a more comprehensive conceptual orientation.

The GRanges class Instances of GRanges are used to specify genomic coordinates. Suppose we wish to represent two D. melanogaster genes. The first is located on the positive strand of chromosome 3R, from position 19967117 to 19973212. The second is on the minus strand of the X chromosome, with ‘left-most’ base at 18962306, and right-most base at 18962925. The coordinates are 1-based (i.e., the first nucleotide on a chromosome is numbered 1, rather than 0), left-most (i.e., reads on the minus strand are defined to ‘start’ at the left-most coordinate, rather than the 5’ coordinate), and closed (the start and end coordinates are included in the range; a range with identical start and end coordinates has width 1, a 0-width range is represented by the special construct where the end coordinate is one less than the start coordinate). A complete definition of these genes as GRanges is:

Table 1: Selected Bioconductor packages for representing and manipulating ranges, strings, and other data structures.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRanges</td>
<td>Defines important classes (e.g., IRanges, Rle) and methods (e.g., findOverlaps, countOverlaps) for representing and manipulating ranges of consecutive values. Also introduces DataFrame, SimpleList and other classes tailored to representing very large data.</td>
</tr>
<tr>
<td>GenomicRanges</td>
<td>Range-based classes tailored to sequence representation (e.g., GRanges, GRangesList), with information about strand and sequence name.</td>
</tr>
<tr>
<td>GenomicFeatures</td>
<td>Foundation for manipulating data bases of genomic ranges, e.g., representing coordinates and organization of exons and transcripts of known genes.</td>
</tr>
</tbody>
</table>
genes <- GRanges(seqnames=c("chr3R", "chrX"),
 ranges=IRanges(
    start=c(19967117, 18962306),
    end = c(19973212, 18962925)),
 strand=c("+", "-"),
 seqlengths=c(chr3R=27905053, chrX=22422827))

The components of a GRanges object are defined as vectors, e.g., of seqnames, much as one would define a data.frame. The start and end coordinates are grouped into an IRanges instance. The optional seqlengths argument specifies the maximum size of each sequence, in this case the lengths of chromosomes 3R and X in the ‘dm2’ build of the D. melanogaster genome. This data is displayed as

```
# GRanges with 2 ranges and 0 metadata columns:
#  seqnames ranges strand
#   chr3R [19967117, 19973212] +
#   chrX [18962306, 18962925] -
#---
# seqlengths:
#  chr3R  chrX
#  27905053 22422827
```

The GRanges class has many useful methods defined on it. Consult the help page `?GRanges` and package vignettes `vignette(package="GenomicRanges")` for a comprehensive introduction. A GRanges instance can be subset, with accessors for getting and updating information.

```
# GRanges with 1 range and 0 metadata columns:
#  seqnames ranges strand
#   chrX [18962306, 18962925] -
#---
# seqlengths:
#  chr3R  chrX
#  27905053 22422827
```

```
strand(genes)
# factor-Rle of length 2 with 2 runs
#  Lengths: 1 1
#  Values : + -
#  Levels(3): + - *
```

```
width(genes)
# [1] 6096  620
```

```
length(genes)
# [1] 2
```
```
names(genes) <- c("FBgn0039155", "FBgn0085359")
genes  # now with names
## GRanges with 2 ranges and 0 metadata columns:
## seqnames ranges strand
## <Rle> <IRanges> <Rle>
## FBgn0039155 chr3R [19967117, 19973212] +
## FBgn0085359 chrX [18962306, 18962925] -
## ---
## seqlengths:
## chr3R chrX
## 27905053 22422827
```

`strand` returns the strand information in a compact representation called a *run-length encoding*. The ‘names’ could have been specified when the instance was constructed; once named, the `GRanges` instance can be subset by name like a regular vector.

As the `GRanges` function suggests, the `GRanges` class extends the `IRanges` class by adding information about `seqnames`, `strand`, and other information particularly relevant to representing ranges that are on genomes. The `IRanges` class and related data structures (e.g., `RangedData`) are meant as a more general description of ranges defined in an arbitrary space. Many methods implemented on the `GRanges` class are ‘aware’ of the consequences of genomic location, for instance treating ranges on the minus strand differently (reflecting the 5’ orientation imposed by DNA) from ranges on the plus strand.

**Operations on ranges** The `GRanges` class has many useful methods. We use `IRanges` to illustrate these operations to avoid complexities associated with strand and seqnames, but the operations are comparable on `GRanges`. We begin with a simple set of ranges:

```
ir <- IRanges(start=c(7, 9, 12, 14, 22:24),
             end=c(15, 11, 12, 18, 26, 27, 28))
```

These and some common operations are illustrated in the upper panel of Figure 1 and summarized in Table 2.

Methods on ranges can be grouped as follows:

**Intra-range** methods act on each range independently. These include `flank`, `narrow`, `reflect`, `resize`, `restrict`, and `shift`, among others. An illustration is `shift`, which translates each range by the amount specified by the `shift` argument. Positive values shift to the right, negative to the left; `shift` can be a vector, with each element of the vector shifting the corresponding element of the `IRanges` instance. Here we shift all ranges to the right by 5, with the result illustrated in the middle panel of Figure 1.
shift(ir, 5)
## IRanges of length 7
## start end width
## [1] 12 20 9
## [2] 14 16 3
## [3] 17 17 1
## [5] 27 31 5
## [6] 28 32 5
## [7] 29 33 5

Inter-range methods act on the collection of ranges as a whole. These include disjoin, reduce, gaps, and range. An illustration is reduce, which reduces overlapping ranges into a single range, as illustrated in the lower panel of Figure 1.

reduce(ir)
## IRanges of length 2
## start end width
## [1] 7 18 12
## [2] 22 28 7

coverage is an inter-range operation that calculates how many ranges overlap individual positions. Rather than returning ranges, coverage returns a compressed representation (run-length encoding)

cvg <- coverage(ir)
cvg
## integer-Rle of length 28 with 12 runs
## Lengths: 6 2 4 1 2 3 1 1 3 1 1
## Values : 0 1 2 1 2 1 0 1 2 3 2 1
## plot(as.integer(cvg), type="s", xlab="Coordinate", ylab="Depth of coverage")

The run-length encoding can be interpreted as ‘a run of length 6 of nucleotides covered by 0 ranges, followed by a run of length 2 of nucleotides covered by 1 range...’.

Between methods act on two (or sometimes more) IRanges instances. These include intersect, setdiff, union, pintersect, psetdiff, and punion.

The countOverlaps and findOverlaps functions operate on two sets of ranges. countOverlaps takes its first argument (the query) and determines how many of the ranges in the second argument (the subject) each overlaps. The result is an integer vector with one element for each member of query. findOverlaps performs a similar operation but returns a more general matrix-like structure that identifies each pair of query/subject overlaps. Both arguments allow some flexibility in the definition of ‘overlap’.

Adding mcols and metadata The GRanges class (actually, most of the data structures defined or extending those in the IRanges package) has two additional very useful data components. The mcols function allows information on each range to be stored and manipulated (e.g., subset) along with the GRanges instance. The element metadata is represented as a DataFrame, defined in IRanges and acting like a standard R data.frame but with the ability to hold more complicated data structures as columns (and with element metadata of its own, providing an enhanced alternative to the Biobase class AnnotatedDataFrame).

mcols(genes) <- DataFrame(EntrezId=c("42865", "2768869"),
Symbol=c("kal-1", "CG34330"))

metadata allows addition of information to the entire object. The information is in the form of a list; any data can be provided.

metadata(genes) <- list(CreatedBy="A. User", Date=date())
Table 2: Common operations on IRanges, GRanges and GRangesList.

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessors</td>
<td>start, end, width</td>
<td>Get or set the starts, ends and widths</td>
</tr>
<tr>
<td></td>
<td>names</td>
<td>Get or set the names</td>
</tr>
<tr>
<td></td>
<td>mcols, metadata</td>
<td>Get or set metadata on elements or object</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>Number of ranges in the vector</td>
</tr>
<tr>
<td></td>
<td>range</td>
<td>Range formed from min start and max end</td>
</tr>
<tr>
<td>Ordering</td>
<td>&lt;, &lt;=, &gt;, &gt;=, ==, !=</td>
<td>Compare ranges, ordering by start then width</td>
</tr>
<tr>
<td></td>
<td>sort, order, rank</td>
<td>Sort by the ordering</td>
</tr>
<tr>
<td></td>
<td>duplicated</td>
<td>Find ranges with multiple instances</td>
</tr>
<tr>
<td></td>
<td>unique</td>
<td>Find unique instances, removing duplicates</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>r + x, r - x, r * x</td>
<td>Shrink or expand ranges r by number x</td>
</tr>
<tr>
<td></td>
<td>shift</td>
<td>Move the ranges by specified amount</td>
</tr>
<tr>
<td></td>
<td>resize</td>
<td>Change width, anchoring on start, end or mid</td>
</tr>
<tr>
<td></td>
<td>distance</td>
<td>Separation between ranges (closest endpoints)</td>
</tr>
<tr>
<td></td>
<td>restrict</td>
<td>Clamp ranges to within some start and end</td>
</tr>
<tr>
<td></td>
<td>flank</td>
<td>Generate adjacent regions on start or end</td>
</tr>
<tr>
<td>Set operations</td>
<td>reduce</td>
<td>Merge overlapping and adjacent ranges</td>
</tr>
<tr>
<td></td>
<td>intersect, union, setdiff</td>
<td>Set operations on reduced ranges</td>
</tr>
<tr>
<td></td>
<td>pintersect, punion, psetdiff</td>
<td>Parallel set operations, on each x[i], y[i]</td>
</tr>
<tr>
<td></td>
<td>gaps, pgap</td>
<td>Find regions not covered by reduced ranges</td>
</tr>
<tr>
<td></td>
<td>disjoin</td>
<td>Ranges formed from union of endpoints</td>
</tr>
<tr>
<td>Overlaps</td>
<td>findOverlaps</td>
<td>Find all overlaps for each x in y</td>
</tr>
<tr>
<td></td>
<td>countOverlaps</td>
<td>Count overlaps of each x range in y</td>
</tr>
<tr>
<td></td>
<td>nearest</td>
<td>Find nearest neighbors (closest endpoints)</td>
</tr>
<tr>
<td></td>
<td>precede, follow</td>
<td>Find nearest y that x precedes or follows</td>
</tr>
<tr>
<td></td>
<td>x %in% y</td>
<td>Find ranges in x that overlap range in y</td>
</tr>
<tr>
<td>Coverage</td>
<td>coverage</td>
<td>Count ranges covering each position</td>
</tr>
<tr>
<td>Extraction</td>
<td>r[i]</td>
<td>Get or set by logical or numeric index</td>
</tr>
<tr>
<td></td>
<td>r[[i]]</td>
<td>Get integer sequence from start[i] to end[i]</td>
</tr>
<tr>
<td></td>
<td>subsetByOverlaps</td>
<td>Subset x for those that overlap in y</td>
</tr>
<tr>
<td></td>
<td>head, tail, rev, rep</td>
<td>Conventional R semantics</td>
</tr>
<tr>
<td>Split, combine</td>
<td>split</td>
<td>Split ranges by a factor into a RangesList</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>Concatenate two or more range objects</td>
</tr>
</tbody>
</table>

The GRangesList class  The GRanges class is extremely useful for representing simple ranges. Some next-generation sequence data and genomic features are more hierarchically structured. A gene may be represented by several exons within it. An aligned read may be represented by discontinuous ranges of alignment to a reference. The GRangesList class represents this type of information. It is a list-like data structure, which each element of the list itself a GRanges instance. The ENSEMBL genes identified earlier can be represented as a GRangesList.

```r
## GRangesList of length 6:
## $84929
## GRanges with 10 ranges and 2 metadata columns:
## senames ranges strand | exon_id exon_name
## <Rle> <IRanges> <Rle> | <integer> <character>
```
## GRanges with 10 ranges and 2 metadata columns:
```
##  seqnames ranges strand  exon_id exon_name
## [1] chr16 [87863629, 87866631]  -  215168 <NA>
## [7] chr16 [87874035, 87874079]  -  215174 <NA>
## [8] chr16 [87874656, 87874761]  -  215175 <NA>
## [9] chr16 [87885330, 87885455]  -  215176 <NA>
## [10] chr16 [87902491, 87903100] -  215177 <NA>
```  
## ...  
## <4 more elements>
## ---  

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

The GRangesList object has methods one would expect for lists (e.g., length, sub-setting). Many of the methods introduced for working with IRanges are also available, with the method applied element-wise.

### Selecting gene sequences

**1.1 Selecting gene sequences**

**Exercise 1**

This exercise uses annotation packages to go from gene identifiers to coding sequences.

1. Map from an informal gene SYMBOL, e.g., BRCA1, to ENTREZID gene identifiers using the org.Hs.eg.db package and the select function, use the TxDb.Hsapiens.UCSC.hg19.knownGene package and a second map to go from ENTREZID to TXNAME.
2. Extract the coding sequence grouped by transcript using the TxDb.Hsapiens.UCSC.hg19.knownGene package and cdsBy function; select just those transcripts we are interested in.
3. Retrieve the nucleotide sequence from the BSgenome.Hsapiens.UCSC.hg19 package using the function extractTranscriptsFromGenome.
4. Verify that the coding sequences are all multiples of 3, and translate from nucleotide to amino acid sequence.

**Solution:** Map from gene SYMBOL to ENTREZID, and from ENTREZID to TXNAME and extract the relevant coding sequence, grouped by transcript

```r
library(org.Hs.eg.db)
egid <- select(org.Hs.eg.db, "BRCA1", "ENTREZID", "SYMBOL")$ENTREZID
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
egToTx <- select(txdb, egid, "TXNAME", "GENEID")

# Warning: 'select' resulted in 1:many mapping between keys and return rows
cdsByTx <- cdsBy(txdb, "tx", use.names=TRUE)[egToTx$TXNAME]
```
Retrieve the sequence

```r
library(BSgenome.Hsapiens.UCSC.hg19)
txx <- extractTranscriptsFromGenome(Hsapiens, cdsByTx)
```

## Warning: 'extractTranscriptsFromGenome' is deprecated.
## Use 'extractTranscriptSeqs' instead.
## See help("Deprecated")

## Warning: 'extractTranscripts' is deprecated.
## Use 'extractTranscriptSeqs' instead.
## See help("Deprecated")

Translate to amino acid sequence

```r
all(width(txx) %% 3 == 0)  # sanity check
## [1] TRUE
translate(txx)              # amino acid sequence
```

1.2 Summarizing overlaps

**Exercise 2**

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 `RNAseqData.HNRNPC.bam.chr14` experiment data package and get the paths to the BAM files it contains.

b. Load the ‘transcript db’ package that contains the coordinates of each exon of the UCSC ‘known genes’ track of hg19.

c. Extract the exon coordinates grouped by gene; the result is an GRangesList object that we will discuss more latter.

d. 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.

e. 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:** Point to BAM files
library(RNAseqData.HNRNPC.bam.chr14)
fls <- RNAseqData.HNRNPC.bam.chr14_BAMFILES

Get the gene model; this could also come from, e.g., a GFF or GTF file.
library(BiocParallel)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
ex <- exonsBy(TxDb.Hsapiens.UCSC.hg19.knownGene, "gene")

Summarize the number of reads overlapping each region of interest
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

References