GenomicRanges for Data and Annotation

Martin T. Morgan[1]

27-28 February 2014

[1] mtmorgan@fhcrc.org

Introduction

Importance of range concepts: conceptually...

- Genomic data and annotation can be represented by ranges
- Biological questions reflect range-based queries

Examples

- How many reads overlap each gene?
- How many reads span splice junctions?
- Where do regulatory elements bind in ChIP-seq experiments?
- Which regulatory elements are closest to differentially expressed genes?
- What sequences are common under discovered regulatory marks?

Where do *GRanges*-like objects come from?

Data

- From BAM files via `readGAlignments` in *GenomicAlignments*
- From BED files via `import` in *rtracklayer*

Annotation

- *rtracklayer* import BED, WIG, GTF, . . . files
- *TxDb.** model organsism gene models; *GenomicFeatures* `makeTranscriptDbFrom*`
- *AnnotationHub* – pre-computed instances from large public resources (later in course)

Key reference

Lawrence et al., 2013, Software for Computing and Annotating Genomic Ranges. PLoS Comput Biol 9(8): e1003118[2]

- Initial developers: Michael Lawrence, Hervé Pagès, Patrick Aboyoun

[2] http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1003118

Ranges

What is a range?

- 'start' and 'end' coordinate vectors
- Closed interval (i.e., include end points)
- Zero-width convention
- Can be 'named'

```
library(IRanges)
eg <- IRanges(start= c(1, 10, 20),
              end  = c(4, 10, 19),
              names= c("A", "B", "C"))
## bigger
start <- floor(runif(10000, 1, 1000))
end <- start + floor(runif(10000, 0, 100))
ir <- IRanges(start, end)
```

'Accessors' and simple manipulation

Accessors

- `start`, `end`, `width`, `names`

'Vector'-like behavior

- `length`, `[`

```
length(ir)

## [1] 10000

ir[1:4]

## IRanges of length 4
##     start end width
## [1]   871 921    51
## [2]   932 975    44
## [3]   916 937    22
## [4]   181 224    44

start(ir[1:4])
```

Operations

1. Intra-range: operate on each range independently, e.g., `shift`
2. Inter-range: operate on several ranges of a single instance, e.g., `reduce`, `coverage`
3. Between-range: operate on two instances, e.g., `findOverlaps`

See table in afternoon lab!

```
ir <- IRanges(start=c(7, 9, 12, 14, 22:24),
              end=c(15, 11, 12, 18, 26, 27, 28))
shift(ir, 1)

## IRanges of length 7
##     start end width
## [1]     8  16     9
## [2]    10  12     3
## [3]    13  13     1
## [4]    15  19     5
## [5]    23  27     5
## [6]    24  28     5
```

IRangesList

- Often useful to group *IRanges* into a list, with each element of the list containing 0 or more *IRanges* instances
- Operations usually work on list element

```
irl <- split(ir, width(ir))
reduce(irl)

## IRangesList of length 4
## $`1`
## IRanges of length 1
##     start end width
## [1]    12  12     1
##
## $`3`
## IRanges of length 1
##     start end width
## [1]     9  11     3
##
```

GRanges

Builds on *IRanges*, *IRangesList*...

- 'seqnames' (e.g., chromosome) and 'strand'
- (optional) 'seqlengths' for genome information
- (optional) 'mcols' for 'metadata' data frame on each range

```
library(GenomicRanges)
genes <- GRanges(seqnames=c("chr3R", "chrX"),
    ranges=IRanges(
      start=c(19967117, 18962306),
      end  =c(19973212, 18962925),
      names=c("FBgn0039155", "FBgn0085359")),
    strand=c("+", "-"),
    seqlengths=c(chr3R=27905053L, chrX=22422827L))
mcols(genes) <-
    DataFrame(EntrezId=c("42865", "2768869"),
              Symbol=c("kal-1", "CG34330"))
```

Coordinates and accessors

Genome coordinates

- 1-based
- 'left-most' – 'start' of ranges on the minus strand are the left-most coordinate, rather than the 5' coordinate.

Accessors

- `seqnames`, `strand`, `seqlengths`, `seqlevels` and like *IRanges*: `start`, `end`, `width`, `names`
- `mcols`; `$` for direct access to metadata

```
width(genes)

## [1] 6096  620

genes$Symbol

## [1] "kal-1"   "CG34330"
```

Operations

- Like *IRanges*, but generally seqnames- and strand-aware
- E.g., `flank` identifies *upstream* (5') region
- E.g., `findOverlaps` checks `seqnames` and `strand`

```
flank(genes, 1000)  ## 5' flanking range

## GRanges with 2 ranges and 2 metadata columns:
##                 seqnames                ranges strand |
##                    <Rle>             <IRanges>  <Rle> | <ch
##   FBgn0039155      chr3R [19966117, 19967116]      +  |
##   FBgn0085359       chrX [18962926, 18963925]      -  |
##                    Symbol
##                 <character>
##   FBgn0039155        kal-1
##   FBgn0085359       CG34330
##   ---
##   seqlengths:
##       chr3R      chrX
```

*List classes

- Often useful to have a list, where all elements of the list are restricted to be of the same type – like *IRangesList*
- Support for common 'atomic' types (*LogicalList*, *IntegerList*, *NumericList*, *CharacterList*, ...) in addition to *IRangesList*, *GRangesList*, ...
- Operations on list elements usually vectorized across elements

```
rl <- splitAsList(1:5, c("A", "B", "A", "B", "B"))
elementLengths(rl)

## A B
## 2 3

log(rl)

## NumericList of length 2
## [["A"]] 0 1.09861228866811
## [["B"]] 0.693147180559945 1.38629436111989 1.6094379124
```

Three advanced concepts

1. *GRanges* extends *IRanges::Vector*, from which it inherits vector-like operations and metadata.
2. **List* data structures are actually vectors + a partitioning, so operations like `unlist`, `relist` and `split` are fast.
3. Many computationally expensive operations, e.g., `findOverlaps` are implemented in C, and are fast.