

Working with DNA strings and ranges

Hervé Pagès

Fred Hutchinson Cancer Research Center

9-10 December, 2010

Introduction

Genomic ranges

Long biological sequences

Full genomes

Resources

Outline

Introduction

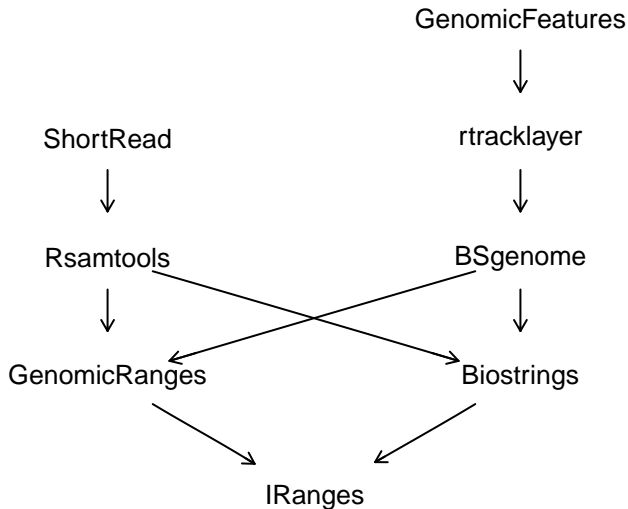
Genomic ranges

Long biological sequences

Full genomes

Resources

Sequence packages



Sequence infrastructure packages

IRanges

Lowest-level containers & tools (*not* Biology-specific)

GenomicRanges

Representation & manipulation of genomic intervals

Biostrings

Representation & manipulation of long biological sequences (DNA, RNA or amino-acids)

BSgenome

High level containers for full genomes

Some of the most important containers

In the *IRanges* package

- ▶ *IRanges*
- ▶ *Rle*
- ▶ *Views*
- ▶ *DataFrame*

In the *GenomicRanges* package

- ▶ *GRanges*
- ▶ *GRangesList*

In the *Biostrings* package

- ▶ *DNASTringSet*

In the *BSgenome* package

- ▶ *BSgenome*

Rle objects

Issue

- ▶ Chromosomes can be hundreds of million of base pairs long, making them hard to manage in computer memory
- ▶ Fortunately, coverage vectors tend to follow an integer step function

Solution

- ▶ Run-length encoding (RLE) is a common compression technique for storing long sequences with lengthy repeats
- ▶ An RLE couples values with run lengths, e.g. the vector 0, 0, 0, 1, 1, 2 would be represented as (3) 0's, (2) 1's, and (1) 2
- ▶ The *IRanges* package uses the *Rle* and *RleList* classes to house coverage vectors

Views objects

Issue

- ▶ Chromosomes can be hundreds of million of base pairs long, making subsequence selection inefficient

Solution

- ▶ Store the original sequence using a pass-by-reference semantic
- ▶ Associate ranges with the sequence to select subsequence
- ▶ Example:
 - ▶ 7007-letter sequence: <<SNIP-3000>>AGATTCA<<SNIP-4000>>
 - ▶ View range: [3001, 3007]
 - ▶ => 7-letter subsequence: AGATTCA

Outline

Introduction

Genomic ranges

Long biological sequences

Full genomes

Resources

Naive representation for intervals with data

Data characteristics

- ▶ Genomic coordinates consist of chromosome, position, and potentially strand information
- ▶ May have additional values, such as GC content or alignment coverage

data.frame approach

```
> chr <- c("chr1", "chr2", "chr1")
> strand <- c("+", "+", "-")
> start <- c(3L, 4L, 1L)
> end <- c(7L, 5L, 3L)
> naive <- data.frame(chr = chr, strand = strand,
+                      start = start, end = end)
```

GRanges construction

GRanges constructor

- ▶ Instances are created using the *GRanges* constructor
- ▶ Starts and ends are wrapped in an *IRanges* constructor
- ▶ Chromosome/contig supplied to *seqnames* argument
- ▶ Underlying sequence lengths can be supplied to *seqlengths* argument

GRanges example

```
> bioc <- GRanges(seqnames = chr,  
+                 ranges = IRanges(start = start, end = end),  
+                 strand = strand,  
+                 seqlengths = c(chr1 = 24, chr2 = 18))
```

GRanges display

GRanges show method

```
> bioc
```

```
GRanges with 3 ranges and 0 elementMetadata values
```

	seqnames	ranges	strand	
	<Rle>	<IRanges>	<Rle>	
[1]	chr1	[3, 7]	+	
[2]	chr2	[4, 5]	+	
[3]	chr1	[1, 3]	-	

```
seqlengths
```

chr1	chr2
24	18

Note

- Optional interval data would appear to the right of | divider

Some common operations on *GRanges* objects

- ▶ `reduce`
- ▶ `gaps`
- ▶ `coverage`
- ▶ `intersect` & `pintersect`
- ▶ `findOverlaps` & `countOverlaps`

Creating a new *GRanges* object

We will use the following *GRanges* objects to illustrate some of those operations

```
> ir <- IRanges(c(1, 8, 14, 15, 19, 34, 40),  
+             width=c(12, 6, 6, 15, 6, 2, 7))  
> strand <- rep(c("+", "-"), c(4, 3))  
> gr <- GRanges(seqnames = "chr1", ranges = ir, strand = strand,  
+             seqlengths = c(chr1 = 50))  
> gr
```

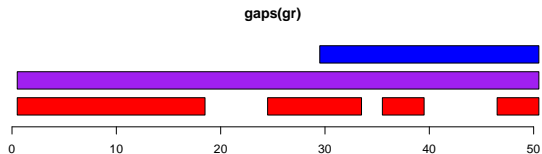
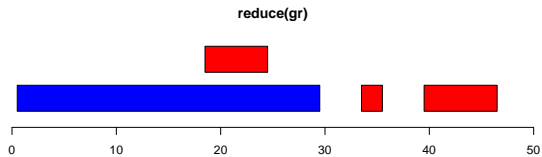
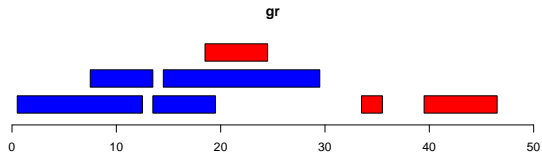
GRanges with 7 ranges and 0 elementMetadata values

	seqnames	ranges	strand	
	<Rle>	<IRanges>	<Rle>	
[1]	chr1	[1, 12]	+	
[2]	chr1	[8, 13]	+	
[3]	chr1	[14, 19]	+	
[4]	chr1	[15, 29]	+	
[5]	chr1	[19, 24]	-	
[6]	chr1	[34, 35]	-	
[7]	chr1	[40, 46]	-	

seqlengths

chr1

50



blue = positive strand, red = negative strand

GRanges subsetting

```
> gr[strand(gr) == "-"]
```

GRanges with 3 ranges and 0 elementMetadata values

	seqnames	ranges	strand	
	<Rle>	<IRanges>	<Rle>	
[1]	chr1	[19, 24]	-	
[2]	chr1	[34, 35]	-	
[3]	chr1	[40, 46]	-	

seqlengths

chr1

50

Finding overlapping ranges

Given 2 objects containing ranges, `findOverlap` (or its variant `countOverlaps`) finds (or counts) the overlaps between the 2 objects

```
> ol <- findOverlaps(gr, reduce(gr))  
> as.matrix(ol)
```

	query	subject
[1,]	1	1
[2,]	2	1
[3,]	3	1
[4,]	4	1
[5,]	5	2
[6,]	6	3
[7,]	7	4

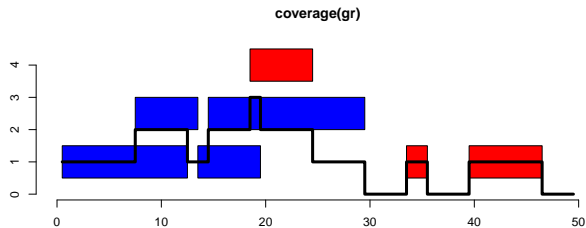
```
> countOverlaps(reduce(gr), gr)
```

```
[1] 4 1 1 1
```

Coverage of the underlying sequence

- ▶ coverage counts the number of ranges over each position in the underlying sequence
- ▶ Subset by strand to get stranded coverage

```
> cvg <- coverage(gr)
```



Outline

Introduction

Genomic ranges

Long biological sequences

Full genomes

Resources

DNA sequences

```
> library(Biostrings)
> data(yeastSEQCHR1)
> c(class(yeastSEQCHR1), nchar(yeastSEQCHR1))

[1] "character" "230208"

> yeast1 <- DNASTring(yeastSEQCHR1)
> yeast1

230208-letter "DNASTring" instance
seq: CCACACCACACCCACACCCACACACC...GGTGTGGTGTGGGTGTGGTGTGTGTGGG

> IUPAC_CODE_MAP

      A      C      G      T      M      R      W      S      Y
"A"    "C"    "G"    "T"    "AC"   "AG"   "AT"   "CG"   "CT"
  K      V      H      D      B      N
"GT"   "ACG"  "ACT"  "AGT"  "CGT"  "ACGT"
```

```
> data(srPhiX174)
> length(srPhiX174)

[1] 1113

> head(srPhiX174, 3)

A DNASTringSet instance of length 3
width seq
[1] 35 GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
[2] 35 GGTGGTTATTATACCGTCAAGGACTGTGTGACTAT
[3] 35 TACCGTCAAGGACTGTGTGACTATTGACGTCCTTC
```

Basic string utilities

Subsequence selection

`subseq`, `Views`

Letter frequencies

`alphabetFrequency`, `dinucleotideFrequency`, `trinucleotideFrequency`,
`oligonucleotideFrequency`, `letterFrequencyInSlidingView`, `uniqueLetters`

Letter consensus

`consensusMatrix`, `consensusString`

Letter transformation

`reverse`, `complement`, `reverseComplement`, `translate`, `chartr`

I/O

`read.DNAStringSet`, `read.RNAStringSet`, `read.AAStringSet`, `read.BStringSet`,
`write.XStringSet`, `save.XStringSet`

String matching/alignment utilities

`matchPDict`

`matchPDict`, `countPDict`, `whichPDict`, `vmatchPDict`, `vcountPDict`, `vwhichPDict`

`vmatchPattern`

`matchPattern`, `countPattern`, `vmatchPattern`, `vcountPattern`

`pairwiseAlignment`

`pairwiseAlignment`, `stringDist`

`matchPWM`

`matchPWM`, `countPWM`

OTHER

`matchLRPatterns`, `trimLRPatterns`, `matchProbePair`, `findPalindromes`,
`findComplementedPalindromes`

Letter frequencies

Single-letter frequencies

```
> alphabetFrequency(yeast1, baseOnly=TRUE)
```

A	C	G	T	other
69830	44643	45765	69970	0

Multi-letter frequencies

```
> dinucleotideFrequency(yeast1)
```

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC
23947	12493	13621	19769	15224	9218	7089	13112	14478	8910
GG	GT	TA	TC	TG	TT				
9438	12938	16181	14021	15617	24151				

```
> head(trinucleotideFrequency(yeast1), 12)
```

AAA	AAC	AAG	AAT	ACA	ACC	ACG	ACT	AGA	AGC	AGG	AGT
8576	4105	4960	6306	3924	2849	2186	3534	4537	2680	2707	3697

Basic transformations

```
> x
  21-letter "DNAString" instance
seq: TCAACGTTGAATAGCGTACCG
> reverseComplement(x)
  21-letter "DNAString" instance
seq: CGGTACGCTATTCAACGTTGA
> translate(x)
  7-letter "AAString" instance
seq: STLNSVP
```


Consensus matrix

```
> snippet <- subseq(head(sort(srPhiX174), 5), 1, 10)
> consensusMatrix(snippet, baseOnly=TRUE)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
A	5	5	1	0	4	2	1	0	1	0
C	0	0	1	0	0	2	0	0	0	0
G	0	0	3	4	0	0	0	4	0	3
T	0	0	0	1	1	1	4	1	4	2
other	0	0	0	0	0	0	0	0	0	0

Consensus string

```
> consensusString(snippet)
```

```
[1] "AAGGAMTGTK"
```

```
> consensusString(snippet, ambiguityMap = "N", threshold = 0.5)
```

```
[1] "AAGGANTGTG"
```

Pattern matching

Count the matches only

```
> data(phiX174Phage)
> genome <- phiX174Phage[["NEB03"]]
> negPhiX174 <- reverseComplement(srPhiX174)
> posCounts <- countPDict(PDict(srPhiX174), genome)
> negCounts <- countPDict(PDict(negPhiX174), genome)
> table(posCounts, negCounts)
```

	negCounts
posCounts	0
0	1030
1	83

Find the locations of the matches

```
> matchPDict(PDict(srPhiX174[posCounts > 0]), genome)
MIndex object of length 83
```

Pairwise alignments

Alignment scores

```
> data(phiX174Phage)
> posScore <- pairwiseAlignment(srPhiX174, genome,
+                               type = "global-local", scoreOnly = TRUE)
> negScore <- pairwiseAlignment(negPhiX174, genome,
+                               type = "global-local", scoreOnly = TRUE)
> cutoff <- max(pmin.int(posScore, negScore))
```

Alignments

```
> pairwiseAlignment(srPhiX174[posScore > cutoff], genome,
+                   type = "global-local")
```

```
Global-Local PairwiseAlignedFixedSubject (1 of 1112)
pattern:  [1] GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
subject: [2750] GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
score: 69.36144
```

Outline

Introduction

Genomic ranges

Long biological sequences

Full genomes

Resources

BSgenome data packages

- ▶ One genome per package
- ▶ Full genome sequences stored in *Biostrings* containers
- ▶ 14 organisms / 24 packages in the current release (BioC 2.7)
- ▶ Most (but not all) packages contain sequences with builtin masks
- ▶ Naming convention: *BSgenome.Organism.Provider.BuildVersion*

Available genomes

Use the `available.genomes` function (from the *BSgenome* software package) to get the list:

```
> library(BSgenome)
> available.genomes()

[1] "BSgenome.Amelliifera.BeeBase.assembly4"
[2] "BSgenome.Amelliifera.UCSC.apiMel2"
[3] "BSgenome.Athaliana.TAIR.01222004"
[4] "BSgenome.Athaliana.TAIR.04232008"
[5] "BSgenome.Btaurus.UCSC.bosTau3"
[6] "BSgenome.Btaurus.UCSC.bosTau4"
[7] "BSgenome.Celegans.UCSC.ce2"
[8] "BSgenome.Celegans.UCSC.ce6"
[9] "BSgenome.Cfamiliaris.UCSC.canFam2"
[10] "BSgenome.Dmelanogaster.UCSC.dm2"
[11] "BSgenome.Dmelanogaster.UCSC.dm3"
[12] "BSgenome.Drerio.UCSC.danRer5"
[13] "BSgenome.Drerio.UCSC.danRer6"
[14] "BSgenome.Ecoli.NCBI.20080805"
[15] "BSgenome.Ggallus.UCSC.galGal3"
[16] "BSgenome.Hsapiens.UCSC.hg17"
[17] "BSgenome.Hsapiens.UCSC.hg18"
[18] "BSgenome.Hsapiens.UCSC.hg19"
[19] "BSgenome.Mmusculus.UCSC.mm8"
[20] "BSgenome.Mmusculus.UCSC.mm9"
[21] "BSgenome.Ptroglodytes.UCSC.panTro2"
[22] "BSgenome.Rnorvegicus.UCSC.rn4"
[23] "BSgenome.Scerevisiae.UCSC.sacCer1"
[24] "BSgenome.Scerevisiae.UCSC.sacCer2"
```

Making your own *BSgenome* data package

- ▶ It's easy to make your own package if your organism is not supported
- ▶ Documented in the BSgenomeForge vignette from the *BSgenome* software package

Working with a genome

```
> library(BSgenome.Celegans.UCSC.ce2)
> Celegans
Worm genome
|
| organism: Caenorhabditis elegans (Worm)
| provider: UCSC
| provider version: ce2
| release date: Mar. 2004
| release name: WormBase v. WS120
|
| single sequences (see '?seqnames'):
|   chrI   chrII   chrIII   chrIV   chrV   chrX   chrM
|
| multiple sequences (see '?mseqnames'):
|   upstream1000 upstream2000 upstream5000
|
| (use the '$' or '[' operator to access a given sequence)
> Celegans$chrI
15080483-letter "DNAString" instance
seq: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAG...CTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
```

Note that a given chromosome sequence is loaded from disk to memory at the time the user makes reference to it (like in the above example), and is unloaded (i.e. the memory is freed) when there are no more references to it.

Working with a genome (continued)

```
> seqlengths(Celegans)
      chrI      chrII      chrIII      chrIV      chrV      chrX      chrM
15080483 15279308 13783313 17493791 20922231 17718849 13794
```

Outline

Introduction

Genomic ranges

Long biological sequences

Full genomes

Resources

Resources

Bioconductor web site

- ▶ `http://bioconductor.org`
- ▶ 'Help' link

Help in *R*

- ▶ `help.start()` to view a help browser.
- ▶ `help(package = "Biostrings")`
- ▶ `?findOverlaps`
- ▶ `browseVignettes("GenomicRanges")`