Working with sequences and intervals

Wolfgang Huber

EMBL Heidelberg, 8 June 2009

IRanges

- Infrastructure to manage and manipulate large sequences and views of their subsequences
- Infrastructure for representing and computing with annotations on sequence regions

Biostrings

- Builds on IRanges infrastructure to represent and manipulate long biological character sequences (DNA / RNA / amino acids)
- Sequence matching and pairwise alignment

BSgenome data packages

- Full genomes stored in Biostrings containers
- Currently 13 organisms supported (Human, Mouse, Worm, Yeast, etc.)
- Facilities for supporting further genomes (BSgenomeForge)

Sequences

In mathematics, a sequence is a function from (a subset of) $\mathbb{Z}$ to an (arbitrary) set $\mathcal{S}$, and can be denoted as $(\ldots, s_0, s_1, s_2, \ldots)$

Atomic vectors in R represent finite sequences of numbers and character strings (with indices from 1 to n). All sequence elements have the same type.

Lists in R represent finite sequences of objects of any type.

Shortcomings:

- Each element is stored explicitly: this can be wasteful for long sequences with repetitive patterns.
- Lists provide no guarantee on uniformity (e.g. of type or size) of their elements.

Sequence containers in the IRanges package

RLE

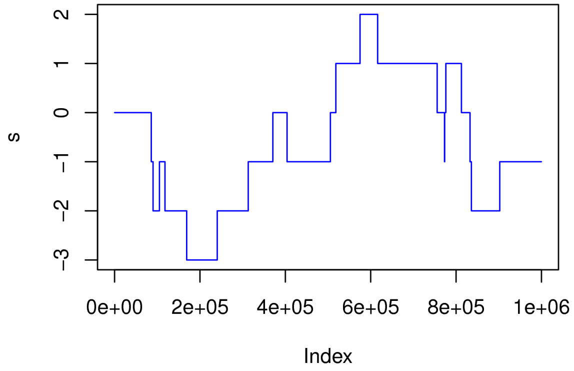
Run length encoding

IRanges

Integer ranges (intervals)

RLE - run length encoding

```
> library("IRanges")
> s = cumsum(round(rnorm(n=1e6, sd=0.12)))
```



RLE - run length encoding

```
> s_rle = Rle(s)
> s_rle

  'numeric' Rle instance of length 1000000 with 22 runs
  Lengths:  85894 4263 14848 13060 50837 71658 72444 57675
  Values :  0 -1 -2 -1 -2 -3 -2 -1 0 -1 ...

> object.size(s)

8000040 bytes

> object.size(s_rle)

1616 bytes
```

RLE objects support usual operations

```
> s1 = Rle(c(0,0,0,0,2,2,2))
> s2 = Rle(c(0,1,1,1,1,0,0))
> s1+s2

  'numeric' Rle instance of length 7 with 4 runs
  Lengths:  1 3 1 2
  Values :  0 1 3 2

> s1[3]

  'numeric' Rle instance of length 1 with 1 run
  Lengths:  1
  Values :  0

> sum(s1)

[1] 6
```

Further sequence operations
(work also with *Rle*)

Sampling in regular intervals

```
> s1 = 1:100
> window(s1, start=5, end=45, delta=5)

[1]  5 10 15 20 25 30 35 40 45
```

Extracting subsequences

```
> seqextract(s1, start=c(10,30,50), width=3)

[1] 10 11 12 30 31 32 50 51 52
```

Achtung

Do not confuse the class *Rle* from the *IRanges* package with the class *rle* defined in R's *base* package - the latter is much less powerful.

IRanges - integer intervals

```
> r = IRanges(
+   start = sample(1000000, 4),
+   width = c(20, 18))

IRanges instance:
      start    end width
[1] 284502 284521    20
[2] 818665 818682    18
[3] 475014 475033    20
[4] 514409 514426    18
```

XSequence class

The *XSequence* virtual class is a general container for storing an ”external sequence”. It inherits from the class *Sequence*, which has a rich interface. The following classes derive from the XSequence class:

- *XRaw*: bytes (stored as `char` values at the C level).
- *XInteger*: integer values (stored as `int`).
- *XNumeric*: numeric values (stored as `double`).
- *XString*: character strings — from Biostrings package.

The purpose of the *X** containers is to provide a *pass by reference* semantic, e. g. in order to avoid the overhead of copying the data when doing computations on a contiguous subsequence.

XSequence class
Efficient computations with long sequences

Extracting a subsequence

```
> xi = XInteger(val=1:20000000)
> system.time({
+    u = subseq(xi, start=10000000, width=3000000)  })

   user  system elapsed
  0.000   0.000   0.001

> system.time({
+    v = xi[10000000:12999999]    })

   user  system elapsed
  0.320   0.012   0.332

> identical(as.integer(u), v)

[1] TRUE
```

Views

Views provide selection of subsequences

The `*Views` classes store a set of views on an arbitrary `Sequence` object, called the *subject* (`XIntegerViews`, `RleViews`, `XStringViews`.)

```
> Views(xi, r)

  Views on a 20000000-integer XInteger subject
subject:        1        2        3 ...    2e+07    2e+07
views:
     start    end width
[1] 284502 284521    20 [284502 284503 ... 284520 284521]
[2] 818665 818682    18 [818665 818666 ... 818681 818682]
[3] 475014 475033    20 [475014 475015 ... 475032 475033]
[4] 514409 514426    18 [514409 514410 ... 514425 514426]
```

Views on subjects of class Rle

```
> Views(s_rle, r)

  Views on a 1000000-length Rle subject

views:
     start    end width
[1] 284502 284521    20 [-2 -2 -2 -2 -2 -2 -2 -2 -2 ...]
[2] 818665 818682    18 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...]
[3] 475014 475033    20 [-1 -1 -1 -1 -1 -1 -1 -1 -1 ...]
[4] 514409 514426    18 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...]
```

What can you do with views?

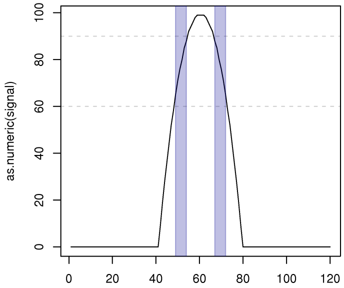
- `[[` extracts an individual object, which is given the same class as the subject
- `restrict`: drop the views that do not overlap with the restriction window, and of the remaining views drop the parts that are outside the window.
- boundaries that are outside the subject are properly handled (see also `trim`).
- `viewSums`, `viewMins`, `viewMaxs`, `viewWhichMins`, `viewWhichMaxs`: fast application of special functions on the views
- `viewApply`: apply any function

Slice

Creating views of the data where they are within a given range

```
> slice(signal, lower = 60, upper = 90)

  Views on a 120-integer XInteger subject
subject: 0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0
views:
    start end width
[1]    49  54     6 [65 71 76 80 85 88]
[2]    67  72     6 [88 85 80 76 71 65]
```

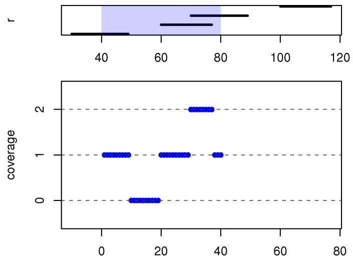


coverage

Given a set of intervals (an `IRanges` or `Views` object), computes how many of them overlap with a given position (or interval)

```
> r = IRanges(start = c(30, 60, 70, 100),
+             width = c(20, 18, 20, 18))
> coverage(r, shift=-40, width=40)
```

```
'integer' Rle instance of length 40 with 5 runs
Lengths:  9 10 10 8 3
Values :  1 0 1 2 1
```



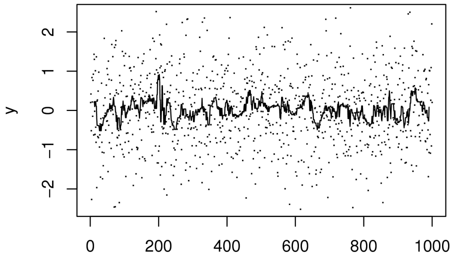
Typical application: `r` a set of enrichment regions from a ChIP-Seq, or of assembled transcripts from an RNA-Seq experiment; `shift` and `width` represent the coordinates of a genomic feature (e. g. annotated gene).

aggregate

combines sequence extraction (`window`) and looping (`sapply`)

Moving median

```
> y = rnorm(1000)
> win = 20
> smy = aggregate(y,
+    start = 1:(length(y)-win+1),
+    width = win,
+    FUN = median)
```



shiftApply

Looping with two sequences, with possible shift

```
> shiftApply

standardGeneric for "shiftApply" defined from package "IRar

function (SHIFT, X, Y, FUN, ..., OFFSET = 0L, simplify = TR
    verbose = FALSE)
standardGeneric("shiftApply")
<environment: 0x3ec3e88>
Methods may be defined for arguments: X, Y
Use  showMethods("shiftApply")  for currently available one
```

sessionInfo()

- R version 2.10.0 Under development (unstable) (2009-06-07 r48726), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=C`, `LC_NUMERIC=C`, `LC_TIME=C`, `LC_COLLATE=C`, `LC_MONETARY=C`, `LC_MESSAGES=it_IT.UTF-8`, `LC_PAPER=it_IT.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=it_IT.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: IRanges 1.3.23, codetools 0.2-2, digest 0.3.1, fortunes 1.3-6, weaver 1.11.0