IRanges

Bioconductor Infrastructure for Sequence Analysis

May 30, 2009

1. Introduction
2. Sequences
3. Ranges
   - Basics
   - Ranges as sets
   - Overlap
4. Data on Ranges
   - Views
   - RangedData

Outline

1. Introduction
2. Sequences
3. Ranges
   - Basics
   - Ranges as sets
   - Overlap
4. Data on Ranges
   - Views
   - RangedData

IRanges

- Supports the manipulation and analysis of:
  - Sequences (ordered collections of elements)
  - Ranges of indices into sequences
  - Data on ranges
- Forms the basis of much of the sequence analysis functionality in Bioconductor
- Emphasis on efficiency in space and time

Outline

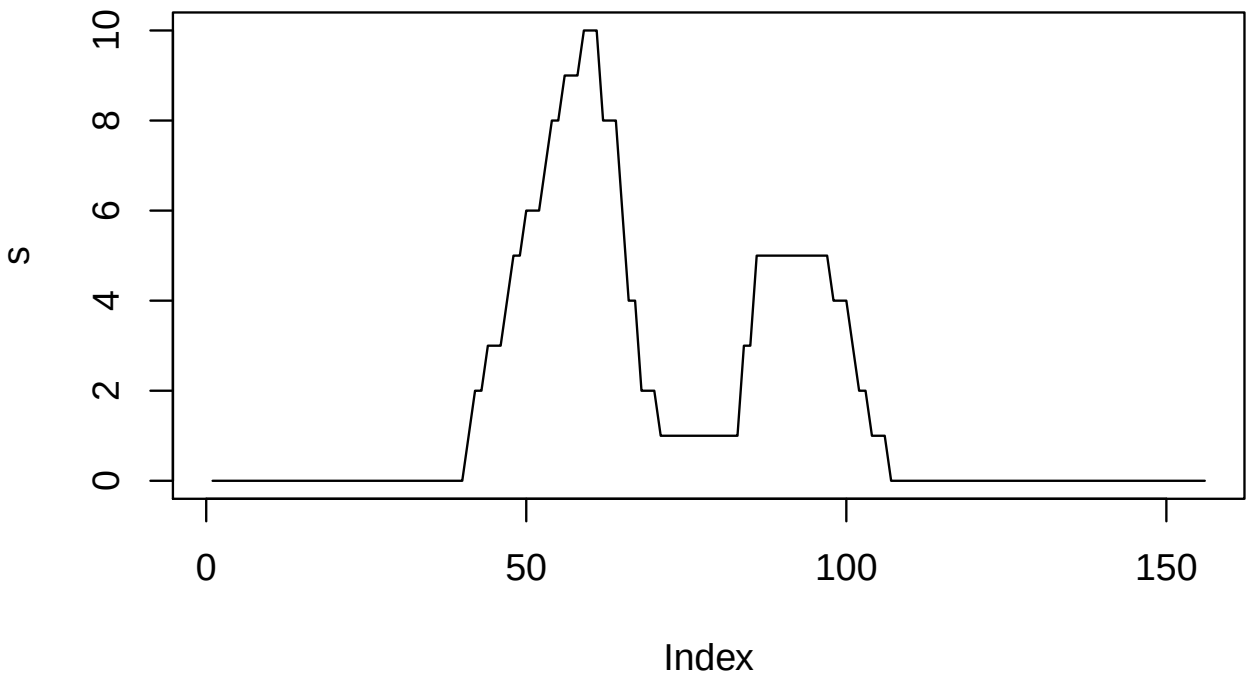
1. Introduction
2. Sequences
3. Ranges
   - Basics
   - Ranges as sets
   - Overlap
4. Data on Ranges
   - Views
   - RangedData

Sequences in IRanges

Almost every object manipulated by *IRanges* is a sequence:

- Atomic sequences (e.g. R vectors)
- Lists
- Data tables (two dimensions)

Example sequence



Run-Length Encoding (RLE)

Our example has many repeated values:

Code

```
> sum(diff(s) == 0)

[1] 133
```

Good candidate for compression by run-length encoding:

Code

```
> sRle <- Rle(s)
> sRle

  'numeric' Rle instance of length 156 with 23 runs
  Lengths:  40 1 2 3 1 2 3 1 2 3 ...
  Values :  0 1 2 3 4 5 6 7 8 9 ...
```

Compression reduces size from 156 to 46.

Rle operations

The *Rle* object like any other sequence/vector:

Basic

```
> sRle > 0 | rev(sRle) > 0

  'logical' Rle instance of length 156 with 3 runs
  Lengths:  40 76 40
  Values :  FALSE TRUE FALSE
```

Summary

```
> sum(sRle > 0)

[1] 66
```

Statistics

```
> cor(sRle, rev(sRle))

[1] 0.5142557
```

EXternal sequences

- Sequences derived from *XSequence* are references
- Memory not copied when containing object is modified
- Example: *XString* in *Biostrings* package, for storing biological sequences efficiently

Outline

1. Introduction
2. Sequences
3. Ranges
   - Basics
   - Ranges as sets
   - Overlap
4. Data on Ranges
   - Views
   - RangedData

Ranges

- Often interested in *consecutive* subsequences
- Consider the alphabet as a sequence:
  - {A, B, C} is a consecutive subsequence
  - The vowels would not be consecutive
- Compact representation: *range* (start and width)
- *Ranges* objects store a sequence of ranges

Creating a Ranges object

The *IRanges* class is a simple *Ranges* implementation.

Code

```
> ir <- IRanges(c(1, 8, 14, 15, 19,
+     34, 40), width = c(12, 6, 6,
+     15, 6, 2, 7))
```

Basic Ranges manipulation

Accessors

```
> start(ir)
[1]  1  8 14 15 19 34 40
> end(ir)
[1] 12 13 19 29 24 35 46
> width(ir)
[1] 12  6  6 15  6  2  7
```

Basic Ranges manipulation

Subsetting

```
> ir[1:5]

IRanges instance:
    start end width
[1]     1  12    12
[2]     8  13     6
[3]    14  19     6
[4]    15  29    15
[5]    19  24     6
```

Normalizing ranges

- *Ranges* can represent a set of integers
- *NormalIRanges* formalizes this, with a compact, normalized representation
- `reduce` normalizes ranges

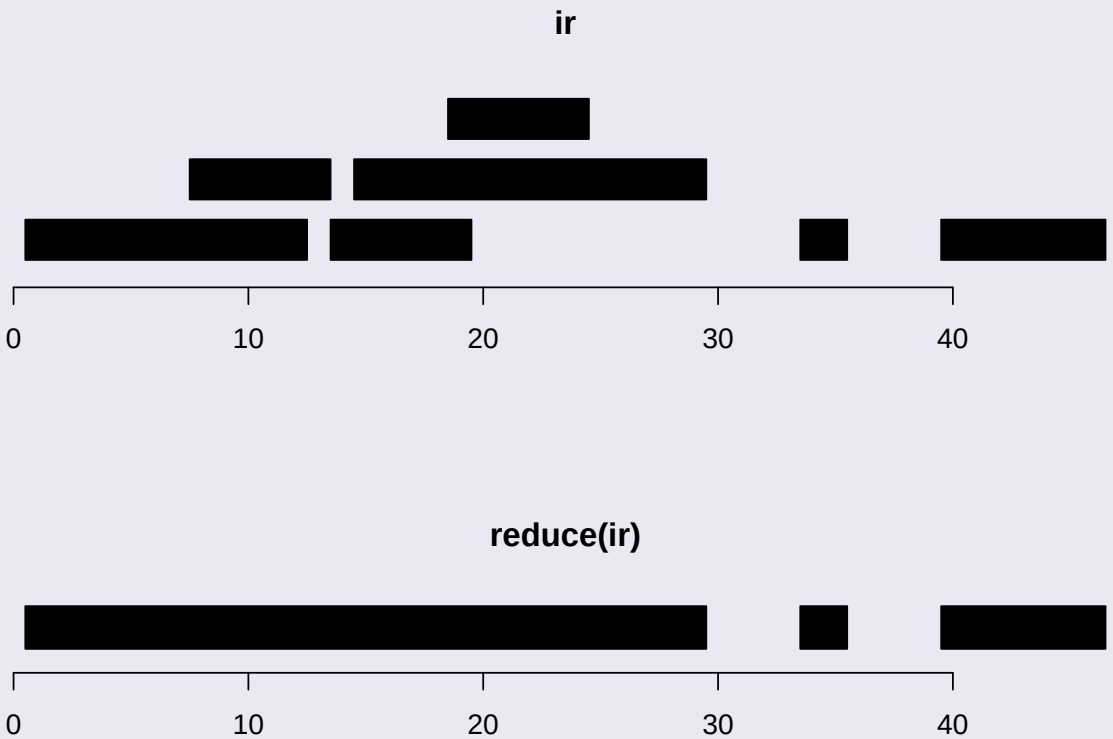
Code

```
> reduce(ir)
```

Normalizing ranges

Code

```
> reduce(ir)
```



Set operations

- *Ranges* as set of integers: `intersect`, `union`, `gaps`, `setdiff`
- Each range as integer set, in parallel: `pintersect`, `punion`, `pgap`, `psetdiff`

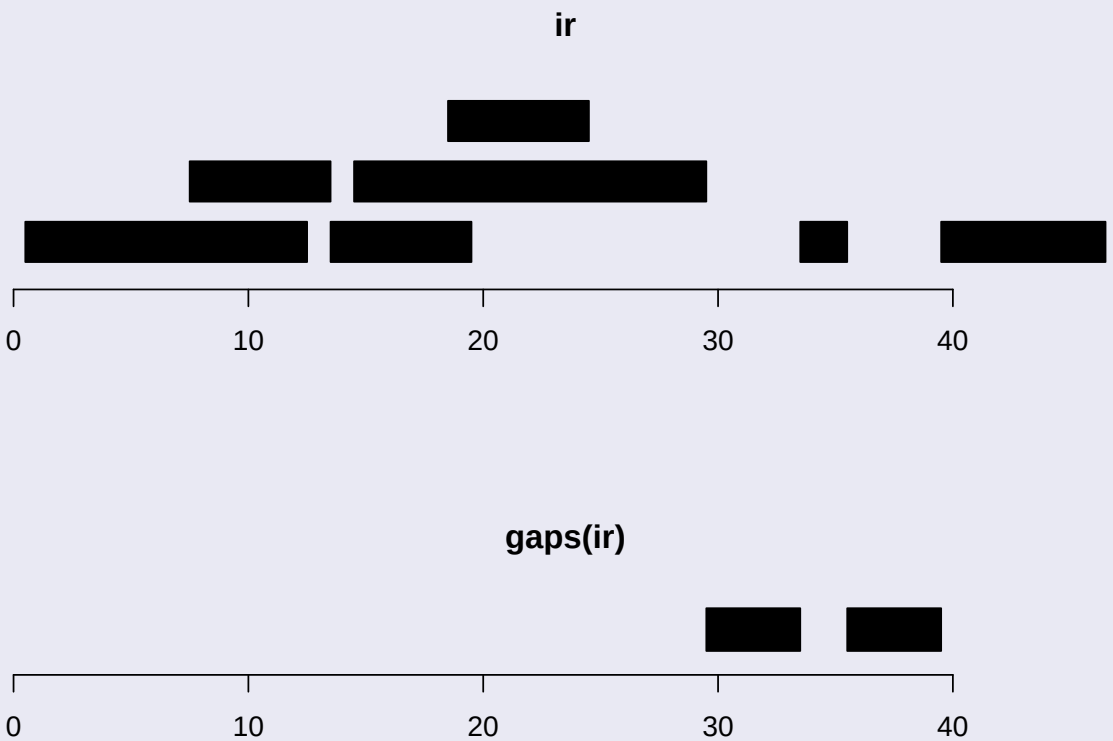
Example: `gaps`

```
> gaps(ir)
```

Set operations

Example: gaps

```
> gaps(ir)
```



Disjoining ranges

- Disjoint ranges are non-overlapping
- `disjoin` returns the widest ranges where the overlapping ranges are the same

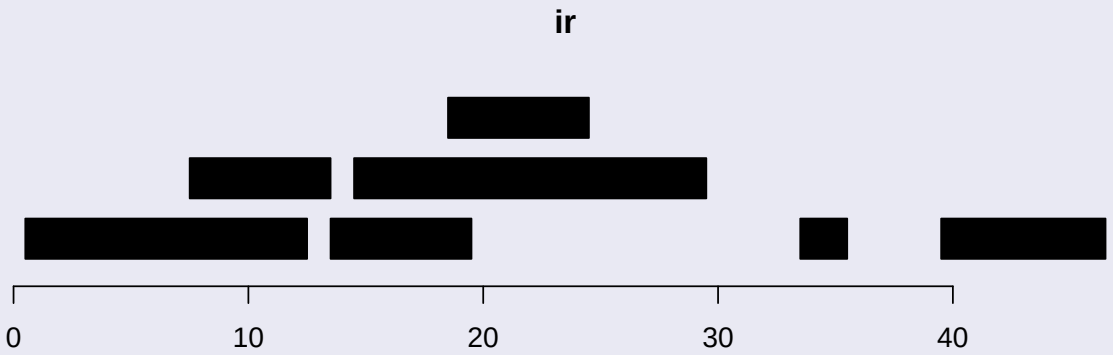
Code

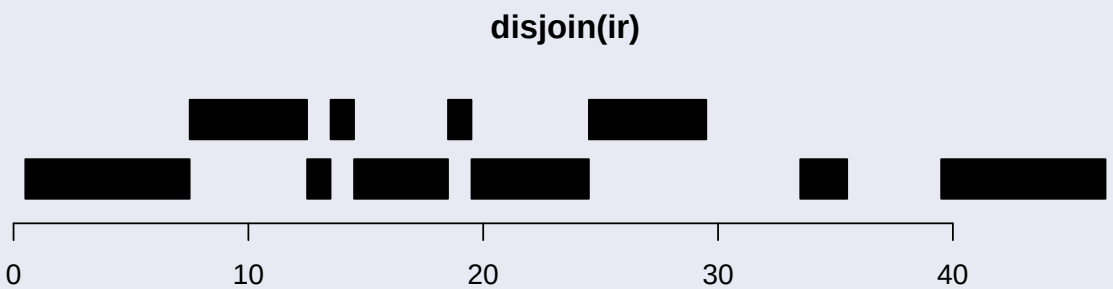
```
> disjoin(ir)
```

Disjoining ranges

Code

```
> disjoin(ir)
```





Overlap detection

- `overlap` detects overlaps between two *Ranges* objects
- Uses interval tree for efficiency

Code

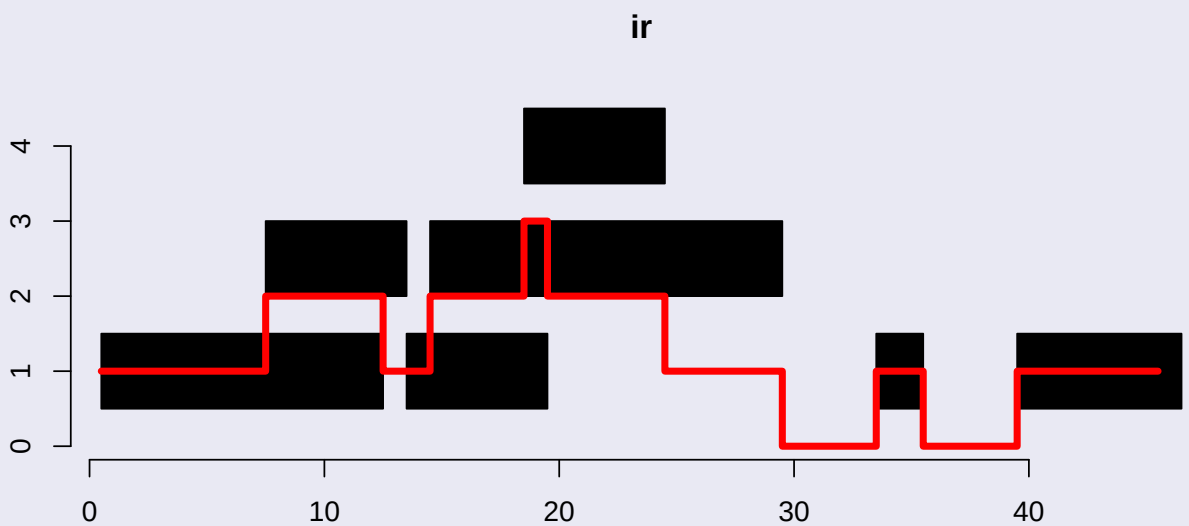
```
> ol <- overlap(reduce(ir), ir)
> as.matrix(ol)

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

Counting overlapping Ranges

`coverage` counts number of ranges over each position

Code

```
> cov <- coverage(ir)
```



Finding nearest neighbors

- `nearest` finds the nearest neighbor ranges (overlapping is zero distance)
- `precede`, `follow` find non-overlapping nearest neighbors on specific side

Outline

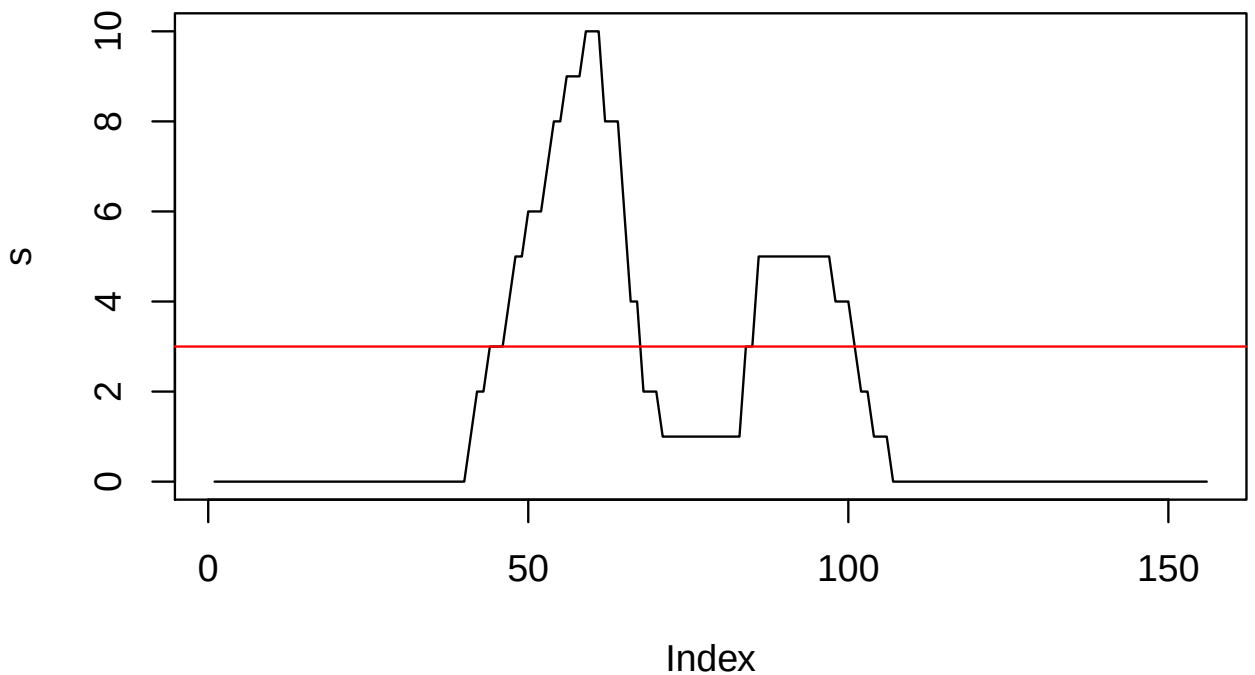
1. Introduction
2. Sequences
3. Ranges
   - Basics
   - Ranges as sets
   - Overlap
4. Data on Ranges
   - Views
   - RangedData

Views

- Associates a *Ranges* object with a sequence
- Sequences can be *Rle* or (in Biostrings) *XString*
- Extends *Ranges*, so supports the same operations

Slicing a Sequence into Views

Goal: find regions above cutoff of 3



Slicing a Sequence into Views

Goal: find regions above cutoff of 3

Using Rle

```
> Views(sRle, as(sRle > 3, "IRanges"))

  Views on a 156-length Rle subject

views:
     start end width
[1]     47  67    21 [ 4  5  5  6 ...]
[2]     86 100    15 [5 5 5 5 5 5 ...]
```

Convenience

```
> sViews <- slice(sRle, 4)
```

Summarizing windows

- Could `sapply` over each window
- Native functions available for common tasks: `viewMins`, `viewMaxs`, `viewSums`, ...

Code

```
> viewSums(sViews)

[1] 150  72

> viewMaxs(sViews)

[1] 10  5
```

RangedData

- Dataset where observations are ranges
- Holds ranges on multiple sequences (e.g. chromosomes)
- Behaves much like *data.frame*
- More during *rtracklayer* talk