Handling genomic data using Bioconductor II: GenomicRanges and GenomicFeatures

Motivating examples

- Genomic “Features” (e.g., genes, exons, CpG islands) on the genome are often represented as intervals, e.g., chromosome, start, end, strand.
  - A common task is to explore the overlaps of two types of features, for example, How CpG islands overlap promoters.
  - Sometimes one wants to obtain the intersect/union of two sets of intervals.
- To obtain a list of genes/exons for an organism.

Without Bioconductor you have to rely on your own scripts for these operations.

Today's topics

- **`GenomicRanges`**: package dealing with genomic intervals (genes, CpG islands, binding sites, etc.)
  - Built on more general package **`IRanges`** for range data.
  - Provide a rich collection of functions for genomic interval operations.
- **`GenomicFeatures`**: package for transcript centric genomic annotations.

`IRanges` package

- "*The IRanges package is designed to represent sequences, ranges representing indices along those sequences, and data related to those ranges*".
  - sequence: ordered finite collection of elements, such as a vector of integers. Not necessarily DNA sequence.
  - Consecutive indices can be represented as a range to save memory and computation, for example, instead of saving c(1,2,3,4,5), just save 1 and 5.

Construct an object of `IRanges`

- Provide start and end indices:

```
> r <- IRanges(start=c(1,3,12,  10), end=c(4,  5, 25, 19))
> r
IRanges of length 4
    start end width
[1]     1   4     4
[2]     3   5     3
[3]    12  25    14
[4]    10  19    10
```

- Or provide start and width of each range:

```
> r <- IRanges(start=c(1,3,12,  10), width=c(4,   3, 14, 10))
> r
IRanges of length 4
    start end width
[1]     1   4     4
[2]     3   5     3
[3]    12  25    14
[4]    10  19    10
```

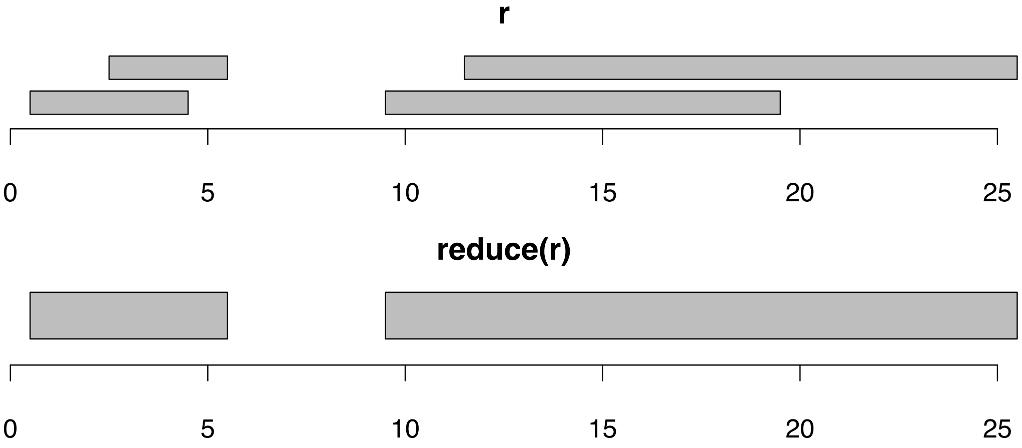
Simple operations of an `IRanges` object

```
> length(r)
[1] 4
> start(r)
[1]  1  3 12 10
> end(r)
[1]  4  5 25 19
> width(r)
[1]  4  3 14 10
> r[1:2]
IRanges of length 2
    start end width
[1]     1   4     4
[2]     3   5     3
> range(r)
IRanges of length 1
    start end width
[1]     1  25    25
```

reduce

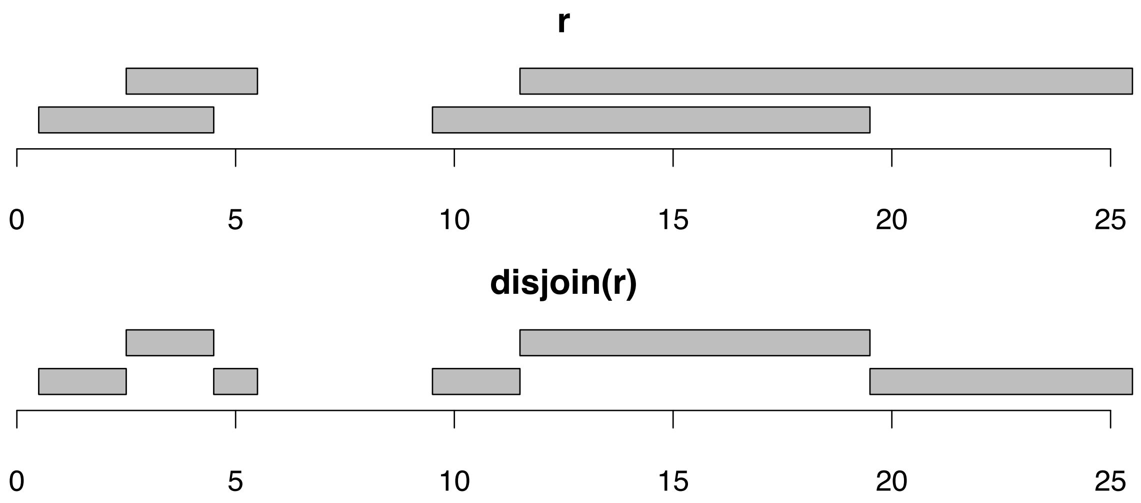
- Merge redundant ranges, and return the minimum non-overlapping ranges covering all the input ranges.

```
> reduce(r)
IRanges of length 2
    start end width
[1]     1   5     5
[2]    10  25    16
```



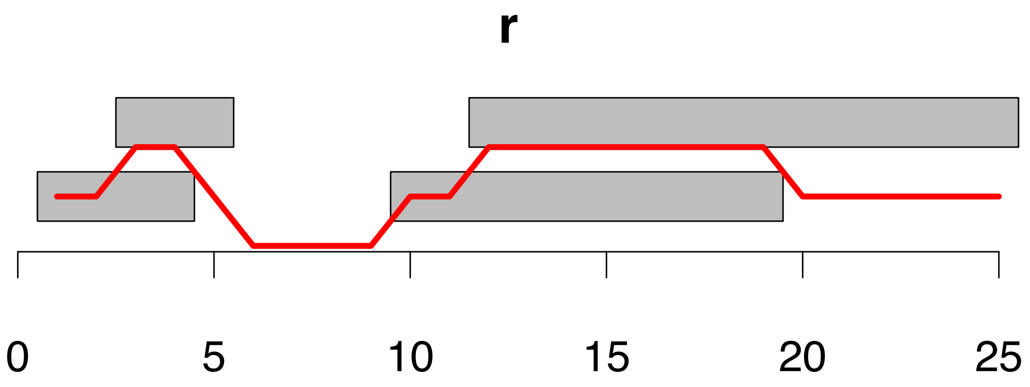
disjoin

- Return a set of non-overlapping ranges satisfying:
  - the union of results is the same as the union of the inputs.
  - for every range in the result, it overlapping pattern with the input is constant.



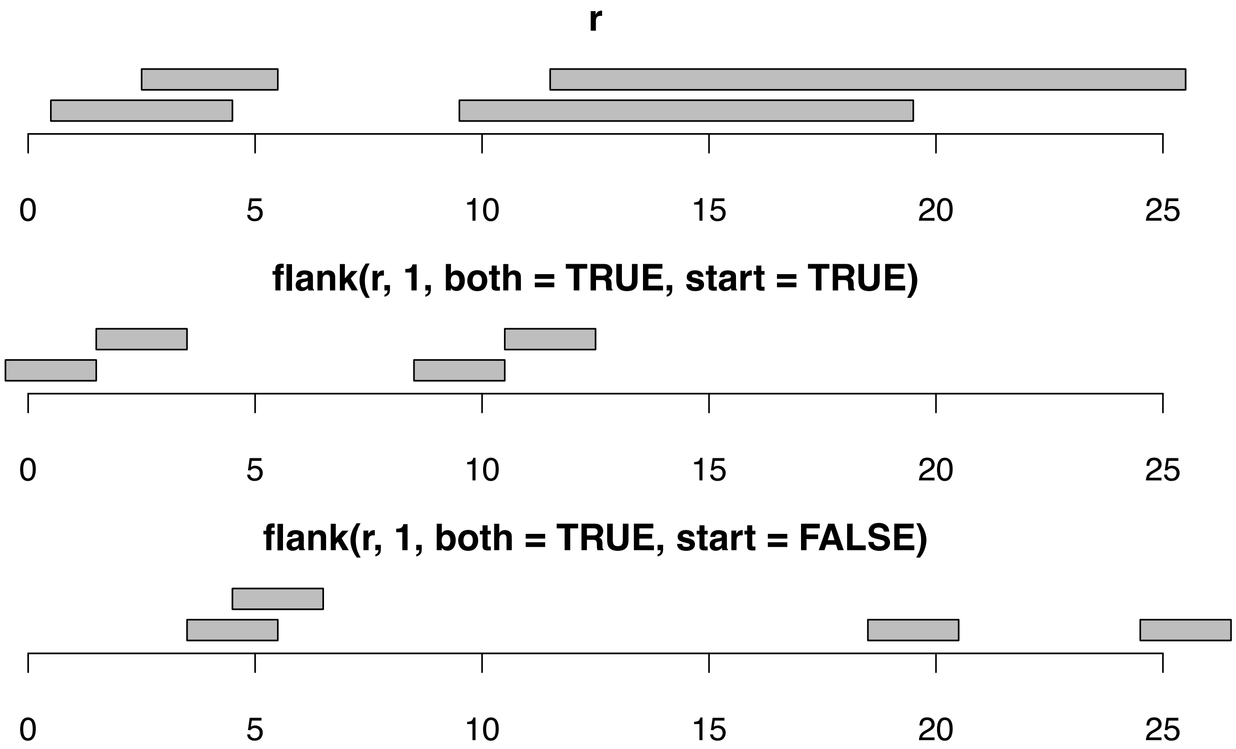
coverage

- Compute the coverage depth by the input ranges of each position.



flank

- Create flanking ranges for each input range.



r

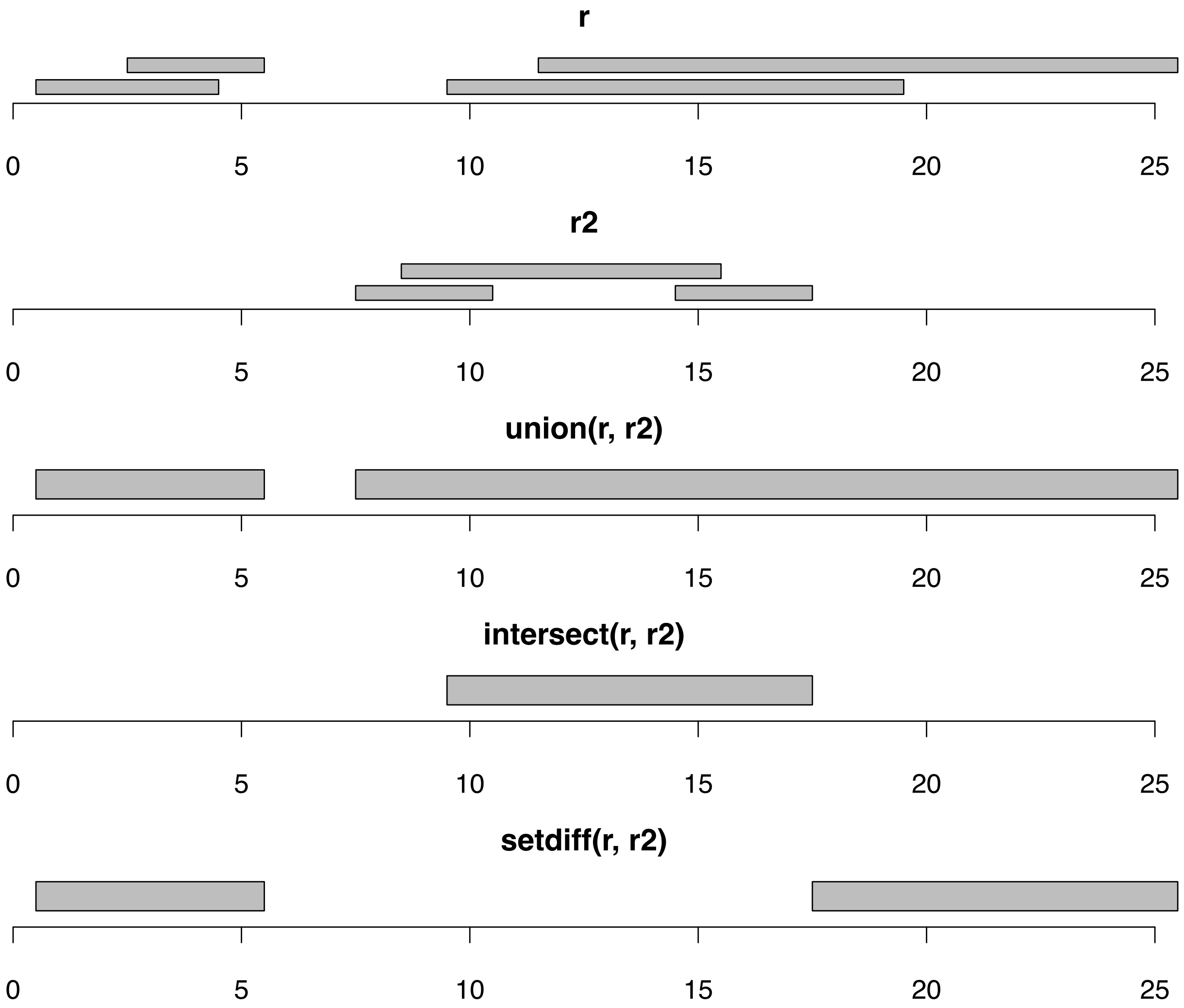
flank(r, 1, both = TRUE, start = TRUE)

flank(r, 1, both = TRUE, start = FALSE)

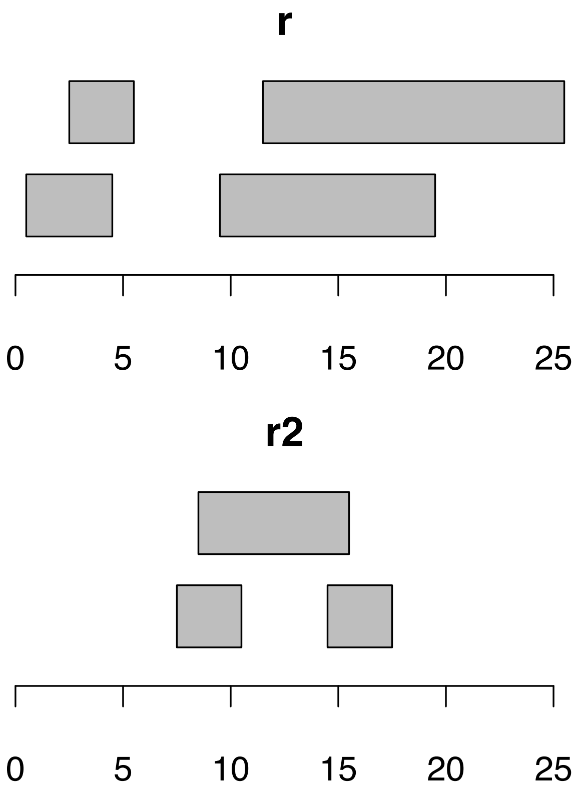
Operations on two `IRanges` objects

- Functions for different set operations of two lists of ranges:
  - union/intersect/setdiff.
  - countOverlaps: for a “query” and a “reference”, count the number of ranges in reference overlapping each range in query.
  - findOverlaps: locating the overlapping ranges in reference for each range in query.

Set operations



overlaps



```
> countOverlaps(r, r2)
[1] 0 0 2 3
> r %over% r2
[1] FALSE FALSE  TRUE  TRUE

> findOverlaps(r, r2)
Hits of length 5
queryLength: 4
subjectLength: 3
  queryHits subjectHits
   <integer>   <integer>
 1          3           2
 2          3           3
 3          4           1
 4          4           2
 5          4           3
```

`Rle`: Run length encoding

- A simple data compression method to represent a long sequence in which consecutive elements often take the same value.
- Instead of saving the whole sequence, it stores the consecutive elements with the same value as a single value and count.

Create `Rle` object

```
> x <- Rle(c(1,1,2,2,2))
> x
'numeric' Rle of length 5 with 2 runs
  Lengths: 2 3
  Values : 1 2
> x <- Rle(values=c(1,2), lengths=c(2,3))
> x
'numeric' Rle of length 5 with 2 runs
  Lengths: 2 3
  Values : 1 2
> as.numeric(x)
[1] 1 1 2 2 2

> x <- Rle(values=c("a","b","c"), lengths=c(2,3,4))
> x
'character' Rle of length 9 with 3 runs
  Lengths:   2   3   4
  Values : "a" "b" "c"
> as.character(x)
[1] "a" "a" "b" "b" "b" "c" "c" "c" "c"
```

Simple operations of `Rle` object

```
> x <- Rle(c(1,1,2,2,2))
> length(x)
[1] 5
> start(x)
[1] 1 3
> end(x)
[1] 2 5
> width(x)
[1] 2 3
> nrun(x)
[1] 2
> runLength(x)
[1] 2 3
```

`GenomicRanges` package

- Designed to represent genomic intervals (genes, CpG islands, binding sites, etc.)
- Based on `IRanges` package and provide support for `BSgenome, GenomicFeatures`, etc.
- Contain three major classes:
  - *GRanges*: single interval range features: a set of genomic features that each has a single start and end locations.
  - *GRangesList*: multiple interval range features: each feature has multiple start/end locations. Ex: a transcript has multiple exons.
  - *GappedAlignments*: gapped alignments.

Create a *GRanges* object

Required fields:

- seqnames: Rle object for sequence name, e.g., the chromosome number.
- ranges: IRanges object for locations.

Other fields: strand, elementMetadata for other information.

```
> gr <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(2, 3)),
+                ranges = IRanges(1:5, end = 6:10),
+                strand = Rle(strand(c("-", "+", "+","-")), c(1,1,2,1)),
+                score = 1:5, GC = seq(1, 0, length = 5))
> gr
GRanges with 5 ranges and 2 elementMetadata values
    seqnames    ranges strand |     score        GC
       <Rle> <IRanges>  <Rle> | <integer> <numeric>
[1]     chr1   [1,  6]      - |         1      1.00
[2]     chr1   [2,  7]      + |         2      0.75
[3]     chr2   [3,  8]      + |         3      0.50
[4]     chr2   [4,  9]      + |         4      0.25
[5]     chr2   [5, 10]      - |         5      0.00

seqlengths
 chr1 chr2
   NA   NA
```

Operate on a GRanges object

```
> length(gr)
[1] 5
> seqnames(gr)
'factor' Rle of length 5 with 2 runs
  Lengths:    2    3
  Values : chr1 chr2
Levels(2): chr1 chr2
> start(gr)
[1] 1 2 3 4 5
> end(gr)
[1]  6  7  8  9 10
> ranges(gr)
IRanges of length 5
    start end width
[1]     1   6     6
[2]     2   7     6
[3]     3   8     6
[4]     4   9     6
[5]     5  10     6
```

```
> strand(gr)
'factor' Rle of length 5 with 3 runs
  Lengths: 1 3 1
  Values : - + -
Levels(3): + - *
> elementMetadata(gr)
DataFrame with 5 rows and 2 columns
      score        GC
  <integer> <numeric>
1         1      1.00
2         2      0.75
3         3      0.50
4         4      0.25
5         5      0.00
```

All other fields (besides seqnames, range and strands) need to be accessed by elementMetadata function, which returns other fields as a data frame.

Subsetting and combining

```
> gr[1:2]
GRanges with 2 ranges and 2 elementMetadata values
    seqnames    ranges strand |     score        GC
       <Rle> <IRanges>  <Rle> | <integer> <numeric>
[1]     chr1    [1, 6]      - |         1      1.00
[2]     chr1    [2, 7]      + |         2      0.75

seqlengths
 chr1 chr2
   NA   NA
> c(gr[1], gr[3])
GRanges with 2 ranges and 2 elementMetadata values
    seqnames    ranges strand |     score        GC
       <Rle> <IRanges>  <Rle> | <integer> <numeric>
[1]     chr1    [1, 6]      - |         1       1.0
[2]     chr2    [3, 8]      + |         3       0.5

seqlengths
 chr1 chr2
   NA   NA
```

Other utility functions

- Inherited from `IRanges` package. Most of the functions working for IRanges also works for GRanges:
  - single range functions: reduce/disjoin/flank/coverage/etc.
  - set operation: intersect/union/setdiff/gap.
  - overlap functions: findOverlap, countOverlap, match, etc.
- The results take into account the chromosome number and strand directions.

```
> coverage(gr)
SimpleRleList of length 2
$chr1
'integer' Rle of length 7 with 3 runs
  Lengths: 1 5 1
  Values : 1 2 1

$chr2
'integer' Rle of length 10 with 6 runs
  Lengths: 2 1 1 4 1 1
  Values : 0 1 2 3 2 1
> reduce(gr)
GRanges with 4 ranges and 0 elementMetadata values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1   [2,  7]      + |
[2]     chr1   [1,  6]      - |
[3]     chr2   [3,  9]      + |
[4]     chr2   [5, 10]      - |
> disjoin(gr)
GRanges with 6 ranges and 0 elementMetadata values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1   [2,  7]      + |
[2]     chr1   [1,  6]      - |
[3]     chr2   [3,  3]      + |
[4]     chr2   [4,  8]      + |
[5]     chr2   [9,  9]      + |
[6]     chr2   [5, 10]      - |

> flank(gr, 2)
GRanges with 5 ranges and 2 elementMetadata values
    seqnames    ranges strand |     score        GC
       <Rle> <IRanges>  <Rle> | <integer> <numeric>
[1]     chr1  [ 7,  8]      - |         1      1.00
[2]     chr1  [ 0,  1]      + |         2      0.75
[3]     chr2  [ 1,  2]      + |         3      0.50
[4]     chr2  [ 2,  3]      + |         4      0.25
[5]     chr2  [11, 12]      - |         5      0.00
```

```
> gr1 <- GRanges(seqnames  = Rle("chr1", 2),
+                ranges=IRanges(start=c(1,10),  end = c(5,15)))
> gr2 <- GRanges(seqnames  = Rle("chr1", 1),
+                ranges = IRanges(start=3,  end = 12))
> union(gr1, gr2)
GRanges with 1 range and 0 elementMetadata  values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1   [1, 15]      * |

> intersect(gr1, gr2)
GRanges with 2 ranges and 0 elementMetadata  values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1  [ 3,  5]      * |
[2]     chr1  [10, 12]      * |

> setdiff(gr1, gr2)
GRanges with 2 ranges and 0 elementMetadata  values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1  [ 1,  2]      * |
[2]     chr1  [13, 15]      * |
```

Overlapping between two GRanges object

- findOverlaps: overlap queries.

```
> findOverlaps(gr1, gr2)
An object of class "RangesMatching"
Slot "matchMatrix":
     query subject
[1,]     1       1
[2,]     2       1

Slot "DIM":
[1] 2 1
```

- %over%: return TRUE/FALSE to indicate if each interval in object 1 overlaps any interval in object 2.

```
> gr1 %over% gr2
[1] TRUE TRUE
```

`GRangesList`: multiple interval range features

- Basically a list of GRanges objects:

```
> GRangesList(gr1, gr2)
GRangesList of length 2
[[1]]
GRanges with 2 ranges and 0 elementMetadata values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1  [ 1,  5]      * |
[2]     chr1  [10, 15]      * |

[[2]]
GRanges with 1 range and 0 elementMetadata values
    seqnames    ranges strand |
       <Rle> <IRanges>  <Rle> |
[1]     chr1   [3, 12]      * |
```

- Subsetting by [[]].
- Support sapply/lapply.

Summary of GenomicRanges

- Provides flexible and efficient functions to operate on the intervals.
- Genomic interval are represented as `GRanges` object, which contains chromosome name in `Rle`, start/end positions as `IRanges` object.
- For second generation sequencing data (will be taught later), each sequence read can be represented as an interval, which makes many operations easier.

GenomicFeatures

- Retrieves and manages different genomic features from public databases (UCSC genome browse and BioMart).
- Provides convenient access for genomic features, compared to manually download and read in text files.

`TranscriptDb` object

- Stores transcript metadata.
- Backed by a SQLite database.
- Three methods to create a new `TranscriptDb` object:
  - `makeTranscriptDbFromUCSC` to download from UCSC Genome browser.
  - `makeTranscriptDbFromBiomart` to download from BioMart.
  - Use a data.frame containing transcript metadata with `makeTranscriptDb` to make a custom database.

makeTranscriptDbFromUCSC

```
> supportedUCSCtables()
                                      track           subtrack
knownGene                        UCSC Genes               <NA>
knownGeneOld3                Old UCSC Genes               <NA>
wgEncodeGencodeManualV3       Gencode Genes    Genecode Manual
wgEncodeGencodeAutoV3         Gencode Genes      Genecode Auto
wgEncodeGencodePolyaV3        Gencode Genes     Genecode PolyA
ccdsGene                               CCDS               <NA>
refGene                        RefSeq Genes               <NA>
xenoRefGene                    Other RefSeq               <NA>
vegaGene                         Vega Genes Vega Protein Genes
vegaPseudoGene                   Vega Genes   Vega Pseudogenes
ensGene                       Ensembl Genes               <NA>
...
```

Creating, saving and loading

```
> txdb=makeTranscriptDbFromUCSC(genom="ce2",tablename="refGene")  ## slow!!!
> txdb
TxDb object:
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: UCSC
# Genome: ce2
# Organism: Caenorhabditis elegans
# UCSC Table: refGene
# Resource URL: http://genome.ucsc.edu/
# Type of Gene ID: Entrez Gene ID
# Full dataset: yes
# miRBase build ID: NA
# transcript_nrow: 50398
# exon_nrow: 153879
# cds_nrow: 131537
# Db created by: GenomicFeatures package from Bioconductor
# Creation time: 2015-09-21 08:56:40 -0400 (Mon, 21 Sep 2015)
# GenomicFeatures version at creation time: 1.20.3
# RSQLite version at creation time: 1.0.0
# DBSCHEMAVERSION: 1.1

> saveDb(txdb, file="ce2_refgenes.sqlite")
> txdb=loadDb("ce2_refgenes.sqlite")
```

Retrieving features

- Retrieve basic features: `transcripts, exons.`

```
> transcripts(txdb)
GRanges object with 50398 ranges and 2 metadata columns:
          seqnames               ranges strand   |     tx_id      tx_name
             <Rle>            <IRanges>  <Rle>   | <integer>  <character>
      [1]     chrI       [11641, 16585]      +   |         1    NM_058259
      [2]     chrI       [15103, 16585]      +   |         2 NM_001306277
      [3]     chrI       [32415, 32435]      +   |         3    NR_049898
      [4]     chrI       [43733, 44676]      +   |         4    NM_058264
      [5]     chrI       [47472, 49414]      +   |         5 NM_001026606
      ...      ...                  ...    ... ...       ...          ...
  [50394]     chrX [17623724, 17627893]      -   |     50394 NM_001029567
  [50395]     chrX [17623724, 17628154]      -   |     50395    NM_171822
  [50396]     chrX [17670503, 17670645]      -   |     50396    NR_072973
  [50397]     chrX [17673384, 17673404]      -   |     50397    NR_072974
  [50398]     chrX [17680821, 17682202]      -   |     50398 NM_001047827
  -------
  seqinfo: 7 sequences (1 circular) from ce2 genome
```

```
> transcripts(txdb, vals=list(tx_chrom="chrI"))

GRanges object with 5258 ranges and 2 metadata columns:
         seqnames               ranges strand   |     tx_id      tx_name
            <Rle>            <IRanges>  <Rle>   | <integer>  <character>
     [1]     chrI       [11641, 16585]      +   |         1    NM_058259
     [2]     chrI       [15103, 16585]      +   |         2 NM_001306277
     [3]     chrI       [32415, 32435]      +   |         3    NR_049898
     [4]     chrI       [43733, 44676]      +   |         4    NM_058264
     [5]     chrI       [47472, 49414]      +   |         5 NM_001026606
     ...      ...                  ...    ... ...       ...          ...
  [5254]     chrI [15071283, 15071432]      -   |      5254    NR_050771
  [5255]     chrI [15075717, 15076404]      -   |      5255    NR_050770
  [5256]     chrI [15076106, 15076404]      -   |      5256    NR_050768
  [5257]     chrI [15078296, 15078629]      -   |      5257    NR_050769
  [5258]     chrI [15078480, 15078629]      -   |      5258    NR_050771
  -------
  seqinfo: 7 sequences (1 circular) from ce2 genome
```

Retrieve by group

- Grouped features functions retrieve features grouped by other features (e.g., genes):
  - `transcriptsBy`, `exonsBy`, `cdsBy`, `intronsByTranscript`, `fiveUTRsByTranscript`, `threeUTRsByTranscript`.

```
> exonsBy(txdb, by=“tx”)

GRangesList object of length 50398:
$1
GRanges object with 3 ranges and 3 metadata columns:
      seqnames         ranges strand |   exon_id   exon_name exon_rank
         <Rle>      <IRanges>  <Rle> | <integer> <character> <integer>
  [1]     chrI [11641, 11689]      + |         1        <NA>         1
  [2]     chrI [14951, 15160]      + |         2        <NA>         2
  [3]     chrI [16473, 16585]      + |         4        <NA>         3

$2
GRanges object with 2 ranges and 3 metadata columns:
      seqnames         ranges strand | exon_id exon_name exon_rank
  [1]     chrI [15103, 15160]      + |       3      <NA>         1
  [2]     chrI [16473, 16585]      + |       4      <NA>         2

$3
GRanges object with 1 range and 3 metadata columns:
      seqnames         ranges strand | exon_id exon_name exon_rank
  [1]     chrI [32415, 32435]      + |       5      <NA>         1

...
<50395 more elements>
-------
seqinfo: 7 sequences (1 circular) from ce2 genome
```

```
> intronsByTranscript(txdb)

GRangesList object of length 50398:
$1
GRanges object with 2 ranges and 0 metadata columns:
      seqnames          ranges strand
         <Rle>       <IRanges>  <Rle>
  [1]     chrI [11690, 14950]      +
  [2]     chrI [15161, 16472]      +

$2
GRanges object with 1 range and 0 metadata columns:
      seqnames          ranges strand
  [1]     chrI [15161, 16472]      +

$3
GRanges object with 0 ranges and 0 metadata columns:
     seqnames ranges strand

...
<50395 more elements>
-------
seqinfo: 7 sequences (1 circular) from ce2 genome
```

Retriving by overlaps

- `transcriptsByOverlaps, exonsByOverlaps, cdsByOverlaps`:
  - return a GRangesList object containing data about transcripts, exons, or coding sequences that overlap genomic coordinates specified by a GRanges object.
  - Useful for, for example, obtain a list of genes overlapping the binding sites of a TF.

```
> gr=GRanges(seqnames = Rle("chrI", 2),
+   ranges=IRanges(start=c(10000,50000), end = c(20000,60000)))
> transcriptsByOverlaps(txdb, gr)

GRanges object with 10 ranges and 2 metadata columns:
       seqnames         ranges strand |     tx_id     tx_name
          <Rle>      <IRanges>  <Rle> | <integer> <character>
   [1]     chrI [11641, 16585]      + |         1   NM_058259
   [2]     chrI [15103, 16585]      + |         2 NM_001306277
   [3]     chrI [49921, 54360]      + |         6   NM_058265
   [4]     chrI [52370, 54360]      + |         7 NM_001306235
   [5]     chrI [ 4221, 10148]      - |      2652   NM_058260
   [6]     chrI [17911, 21127]      - |      2653 NM_001306279
   [7]     chrI [17911, 26643]      - |      2654 NM_001306278
   [8]     chrI [17911, 26778]      - |      2655   NM_058261
   [9]     chrI [17911, 26778]      - |      2656   NM_058262
  [10]     chrI [55337, 63972]      - |      2658   NM_058267
  -------
  seqinfo: 7 sequences (1 circular) from ce2 genome
```

A practical example

- Assume I have a list of TF binding sites in human genome hg19, How to obtain:
  - GC content (%G+%C) of each site.
  - percentage of gene promoters covered by the binding sites.
- Steps:
  1. Load in BSgenome.Hsapiens.UCSC.hg19.
  2. For each site, retrieve its DNA sequence (use Views to speed up).
  3. Use alphabetFrequency to compute GC content.
  4. Create GRanges object to represent the binding sites.
  5. Retrieve gene locations using GenomicFeatures.
  6. Create GRanges to represent all the gene promoters.
  7. Use countOverlaps to analyze the overlap.

biomaRt

- R interface to the BioMart databases (http://www.biomart.org).
- Examples of BioMart databases are Ensembl, Uniprot and HapMap.
- Works similarly to GenomicFeatues, but slower since everything has to be retrieved from internet.
- More flexible: have connections with affy ID and GO annotation, etc.

Review

- We have introduced following useful Bioconductor package: **`GenomicRanges`**, **`GenomicFeatures`**.
- Use a combination of these and `Biostrings/BSgenome`, you can easily achieve most routine analysis works for bioinformatician.
- After class:
  - Review slides and rerun the R codes (on the class webpage).
  - Install `GenomicRanges and GenomicFeatures`.