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
```

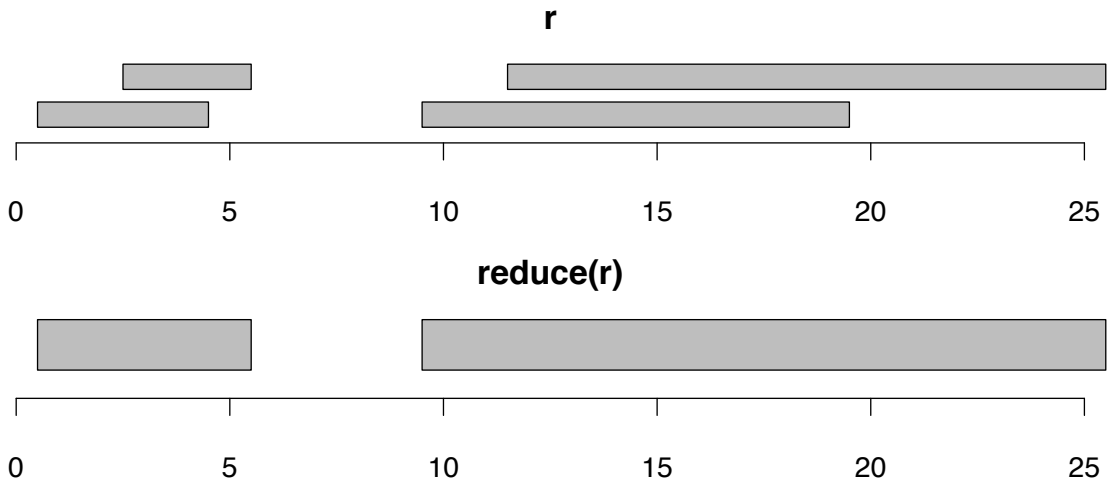
Simple operations of an `IRanges` object

```
> 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

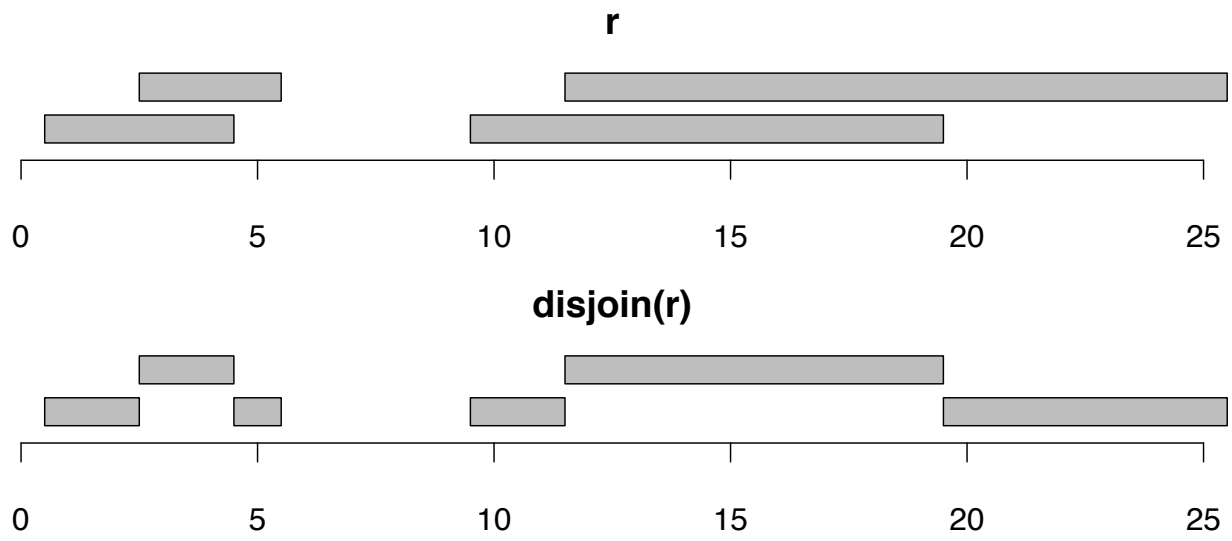
- Merge redundant ranges, and return the minimum non-overlapping ranges covering all the input ranges.

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



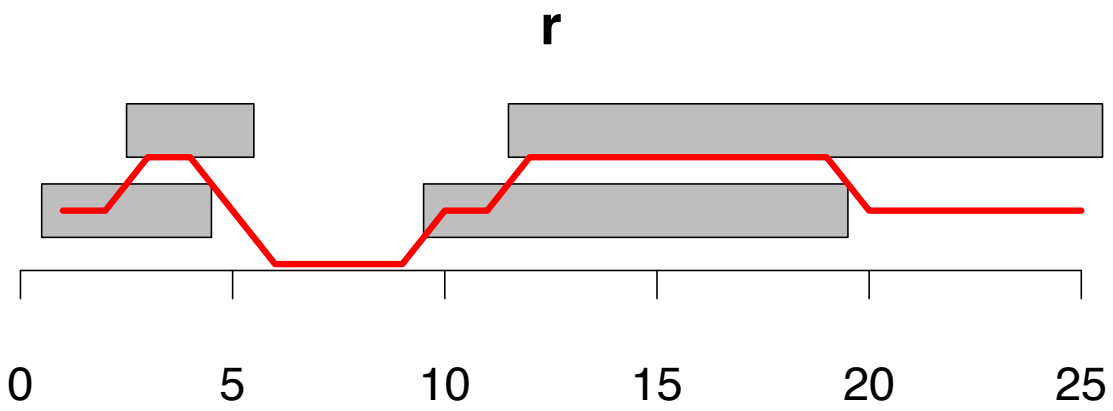
disjoin

- 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 the same.



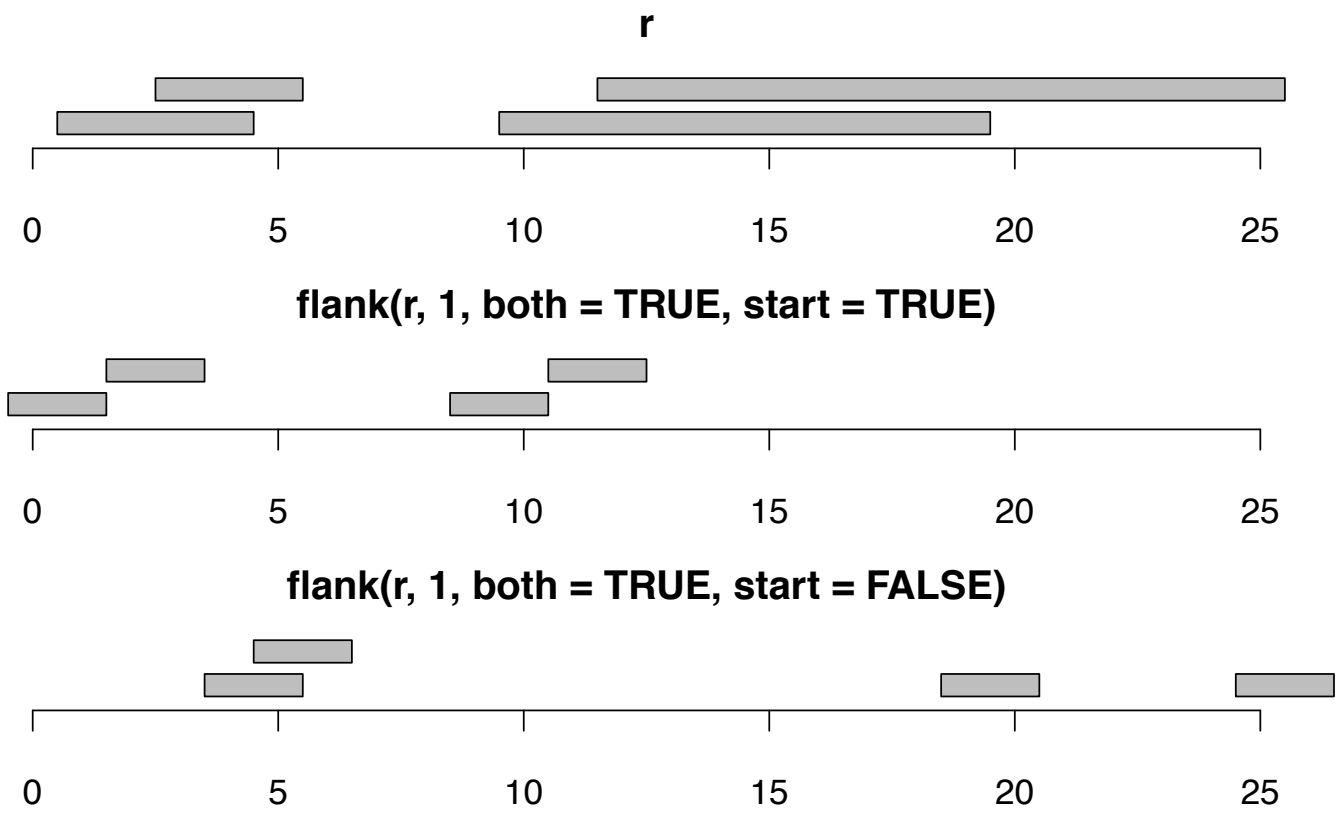
coverage

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



flank

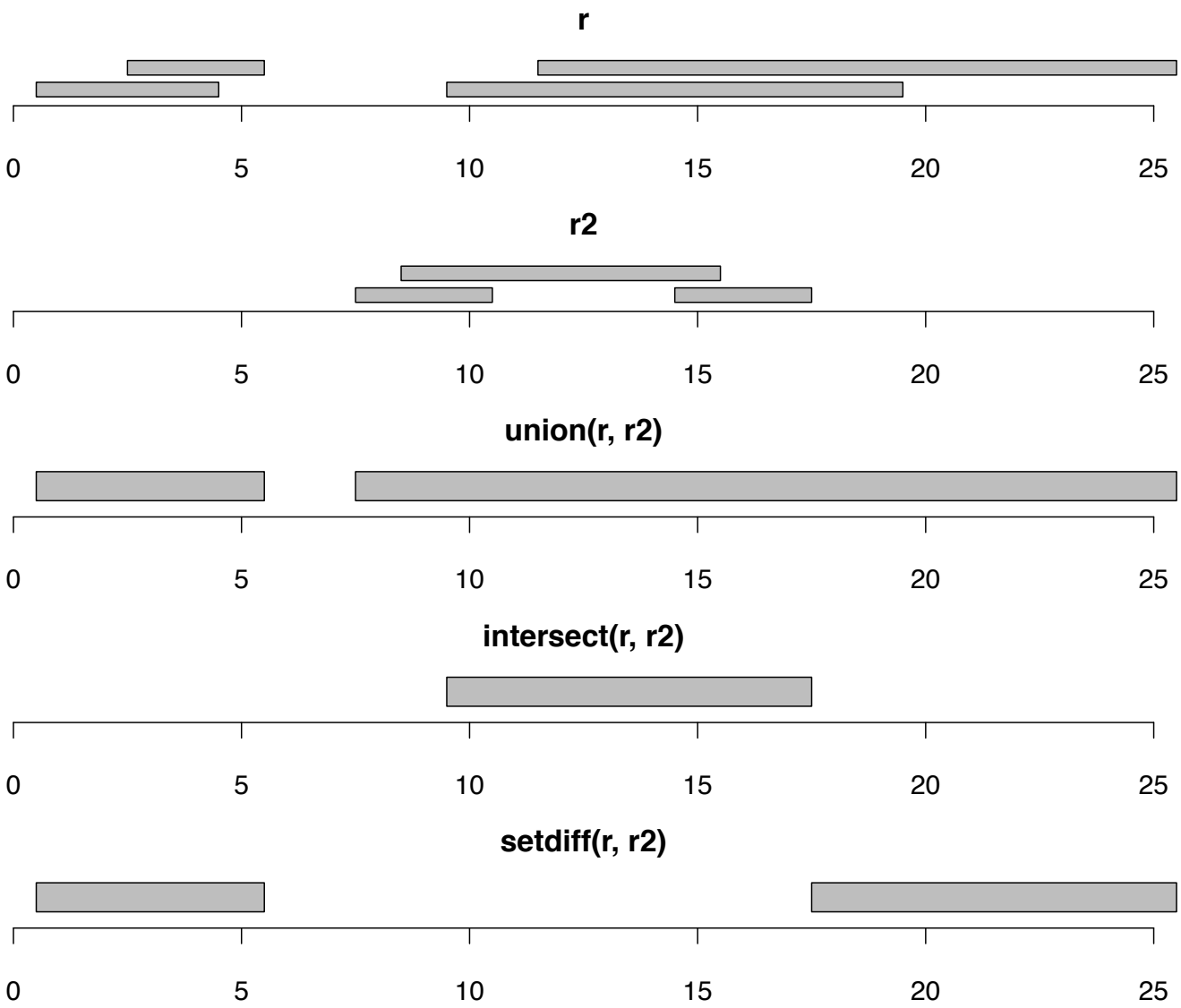
- Create flanking ranges for each input range.



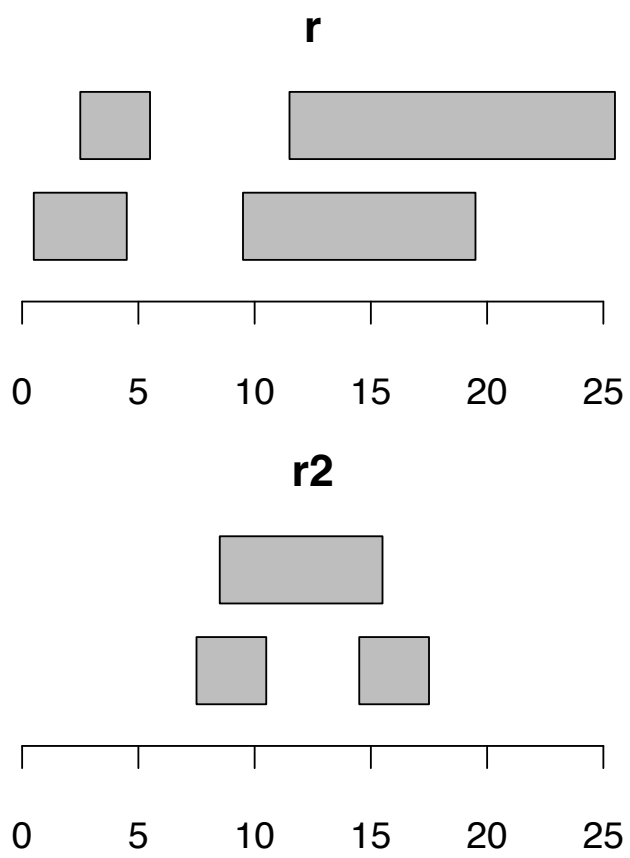
Operations on two `IRanges` objects

- 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 object with 5 ranges and 2 metadata columns:
      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
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

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 object with 5 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [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 DataFrame.

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 consider the chromosome number and strand directions.

```
> coverage(gr)

RleList 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 object with 4 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1       2-7      +
  [2]     chr1       1-6      -
  [3]     chr2       3-9      +
  [4]     chr2      5-10      -
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> disjoin(gr)
GRanges object with 6 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1       2-7      +
  [2]     chr1       1-6      -
  [3]     chr2         3      +
  [4]     chr2       4-8      +
  [5]     chr2         9      +
  [6]     chr2      5-10      -
  -------
  seqinfo: 2 sequences from an unspecified genome;
no seqlengths
```

```
> flank(gr, 2)
GRanges object with 5 ranges and 2 metadata columns:
      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
  -------
  seqinfo: 2 sequences from an unspecified genome; no
seqlengths
```

```
> 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 object with 1 range and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1      1-15      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> intersect(gr1, gr2)
GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1       3-5      *
  [2]     chr1     10-12      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Overlapping between two GRanges object

- findOverlaps: overlap queries.

```
> findOverlaps(gr1, gr2)
Hits object with 2 hits and 0 metadata columns:
      queryHits subjectHits
      <integer>   <integer>
  [1]         1           1
  [2]         2           1
  -------
  queryLength: 2 / subjectLength: 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 object of length 2:
[[1]]
GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1       1-5      *
  [2]     chr1     10-15      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

[[2]]
GRanges object with 1 range and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1      3-12      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

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

Summary of GenomicRanges

- Provides flexible and efficient functions to operate on the intervals on the genome.
- 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 browser and BioMart).
- Provides more convenient access for genomic features, compared to manually download and read in text files.

`TxDb` object

- Stores transcript annotations.
- Backed by a SQLite database.
- Three methods to create a new `TxDb` object:
  - `makeTxDbFromUCSC` to download from UCSC Genome browser.
  - `makeTxDbFromBiomart` to download from BioMart.
  - Use a GFF3 or GTF file containing transcript information with `makeTxDbFromGFF`.
- `makeTxDbPackageFromUCSC` and `makeTxDbPackageFromBiomart` can make annotation packages for later use.

makeTxDbFromUCSC

```
> supportedUCSCtables()
tablename           track              subtrack
1             knownGene      UCSC Genes                  <NA>
2         knownGeneOld8 Old UCSC Genes                    <NA>
3         knownGeneOld7 Old UCSC Genes                    <NA>
4         knownGeneOld6 Old UCSC Genes                    <NA>
5         knownGeneOld4 Old UCSC Genes                    <NA>
6         knownGeneOld3 Old UCSC Genes                    <NA>
7             ccdsGene             CCDS                 <NA>
8           xenoRefGene   Other RefSeq                   <NA>
9             vegaGene      Vega Genes Vega Protein Genes
10      vegaPseudoGene      Vega Genes   Vega Pseudogenes
11            ensGene  Ensembl Genes                    <NA>
...
```

Creating, saving and loading

```
> txdb=makeTxDbFromUCSC(genome="hg19",
                        tablename="knownGene")

Download the knownGene table ... OK
Download the knownToLocusLink table ... OK
Extract the 'transcripts' data frame ... OK
Extract the 'splicings' data frame ... OK
Download and preprocess the 'chrominfo' data frame ... OK
Prepare the 'metadata' data frame ... OK
Make the TxDb object ... OK
```

```
> txdb
TxDb object:
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: UCSC
# Genome: hg19
# Organism: Homo sapiens
# Taxonomy ID: 9606
# UCSC Table: knownGene
# UCSC Track: UCSC Genes
...
# transcript_nrow: 82960
# exon_nrow: 289969
# cds_nrow: 237533
# Db created by: GenomicFeatures package from Bioconductor
...

> saveDb(txdb, file="hg19_knownGenes.sqlite")
> txdb = loadDb("hg19_knownGenes.sqlite")
```

`makeTxDbPackageFromUCSC`

- Directly make a package for the TxDb
- Need to provide some basic information for R package building (maintainer, author, version, etc.)
- Result is an R package in current directory.
- Need to install the package and then it can be used.

```
> makeTxDbPackageFromUCSC(
          maintainer="Hao Wu <hao.wu@emory.edu>",
          author="Hao Wu",
          version="1.0”,
          genome="hg19",
          tablename="knownGene")

Download the knownGene table ... OK
Download the knownToLocusLink table ... OK
Extract the 'transcripts' data frame ... OK
Extract the 'splicings' data frame ... OK
Download and preprocess the 'chrominfo' data frame ... OK
Prepare the 'metadata' data frame ... OK
Make the TxDb object ... OK
Creating package in ./TxDb.Hsapiens.UCSC.hg19.knownGene
```

TxDb object vs. package

- Contain the same information
- A package might be easier to maintain and share.

Retrieving features

- Retrieve basic features: `transcripts, exons.`

```
> transcripts(txdb)
GRanges object with 82960 ranges and 2 metadata columns:
                seqnames            ranges strand |     tx_id     tx_name
                   <Rle>         <IRanges>  <Rle> | <integer> <character>
       [1]           chr1 [ 11874,  14409]      + |         1  uc001aaa.3
       [2]           chr1 [ 11874,  14409]      + |         2  uc010nxq.1
       [3]           chr1 [ 11874,  14409]      + |         3  uc010nxr.1
       [4]           chr1 [ 69091,  70008]      + |         4  uc001aal.1
       [5]           chr1 [321084, 321115]      + |         5  uc001aaq.2
       ...            ...               ...    ... .       ...         ...
   [82956] chrUn_gl000237   [    1,  2686]      - |     82956  uc011mgu.1
   [82957] chrUn_gl000241   [20433, 36875]      - |     82957  uc011mgv.2
   [82958] chrUn_gl000243   [11501, 11530]      + |     82958  uc011mgw.1
   [82959] chrUn_gl000243   [13608, 13637]      + |     82959  uc022brq.1
   [82960] chrUn_gl000247   [ 5787,  5816]      - |     82960  uc022brr.1
   -------
   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
> transcripts(txdb, filter=list(tx_chrom="chr1"))
GRanges object with 7967 ranges and 2 metadata columns:
         seqnames                 ranges strand |     tx_id     tx_name
            <Rle>              <IRanges>  <Rle> | <integer> <character>
     [1]     chr1       [ 11874,  14409]      + |         1  uc001aaa.3
     [2]     chr1       [ 11874,  14409]      + |         2  uc010nxq.1
     [3]     chr1       [ 11874,  14409]      + |         3  uc010nxr.1
     [4]     chr1       [ 69091,  70008]      + |         4  uc001aal.1
     [5]     chr1       [321084, 321115]      + |         5  uc001aaq.2
     ...      ...                    ...    ... .       ...         ...
  [7963]     chr1 [249144203, 249152264]      - |      7963  uc031pta.1
  [7964]     chr1 [249144203, 249152912]      - |      7964  uc001ifb.2
  [7965]     chr1 [249144203, 249153125]      - |      7965  uc010pzr.2
  [7966]     chr1 [249144203, 249153315]      - |      7966  uc001ifc.2
  [7967]     chr1 [249144203, 249153315]      - |      7967  uc001iff.2
  -------
   seqinfo: 298 sequences (2 circular) from hg19 genome
```

```
> exons(txdb)
GRanges object with 289969 ranges and 1 metadata column:
                seqnames          ranges strand |   exon_id
                   <Rle>       <IRanges>  <Rle> | <integer>
       [1]           chr1 [11874, 12227]      + |         1
       [2]           chr1 [12595, 12721]      + |         2
       [3]           chr1 [12613, 12721]      + |         3
       [4]           chr1 [12646, 12697]      + |         4
       [5]           chr1 [13221, 14409]      + |         5
       ...            ...             ...    ... .       ...
  [289965] chrUn_gl000241 [35706, 35859]      - |    289965
  [289966] chrUn_gl000241 [36711, 36875]      - |    289966
  [289967] chrUn_gl000243 [11501, 11530]      + |    289967
  [289968] chrUn_gl000243 [13608, 13637]      + |    289968
  [289969] chrUn_gl000247 [ 5787,  5816]      - |    289969
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome
```

Retrieve by group

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

```
> transcriptsBy(txdb, by="gene")
GRangesList object of length 23459:
$1
GRanges object with 2 ranges and 2 metadata columns:
      seqnames               ranges strand |     tx_id     tx_name
         <Rle>            <IRanges>  <Rle> | <integer> <character>
  [1]    chr19 [58858172, 58864865]      - |     70455  uc002qsd.4
  [2]    chr19 [58859832, 58874214]      - |     70456  uc002qsf.2

$10
GRanges object with 1 range and 2 metadata columns:
      seqnames               ranges strand | tx_id    tx_name
  [1]     chr8 [18248755, 18258723]      + | 31944 uc003wyw.1

$100
GRanges object with 1 range and 2 metadata columns:
      seqnames               ranges strand | tx_id    tx_name
  [1]    chr20 [43248163, 43280376]      - | 72132 uc002xmj.3

...
<23456 more elements>
-------
seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
> exonsBy(txdb, by="gene")
GRangesList object of length 23459:
$1
GRanges object with 15 ranges and 2 metadata columns:
       seqnames               ranges strand |   exon_id   exon_name
          <Rle>            <IRanges>  <Rle> | <integer> <character>
   [1]    chr19 [58858172, 58858395]      - |    250809        <NA>
   [2]    chr19 [58858719, 58859006]      - |    250810        <NA>
   [3]    chr19 [58859832, 58860494]      - |    250811        <NA>
   [4]    chr19 [58860934, 58862017]      - |    250812        <NA>
   [5]    chr19 [58861736, 58862017]      - |    250813        <NA>
   ...      ...                 ...    ... .       ...         …

$10
GRanges object with 2 ranges and 2 metadata columns:
      seqnames               ranges strand | exon_id exon_name
  [1]     chr8 [18248755, 18248855]      + |  113603      <NA>
  [2]     chr8 [18257508, 18258723]      + |  113604      <NA>

...
<23457 more elements>
-------
seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
> intronsByTranscript(txdb)
GRangesList object of length 82960:
$1
GRanges object with 2 ranges and 0 metadata columns:
      seqnames          ranges strand
         <Rle>       <IRanges>  <Rle>
  [1]     chr1 [12228, 12612]      +
  [2]     chr1 [12722, 13220]      +

$2
GRanges object with 2 ranges and 0 metadata columns:
      seqnames          ranges strand
  [1]     chr1 [12228, 12594]      +
  [2]     chr1 [12722, 13402]      +

$3
GRanges object with 2 ranges and 0 metadata columns:
      seqnames          ranges strand
  [1]     chr1 [12228, 12645]      +
  [2]     chr1 [12698, 13220]      +

...
<82957 more elements>
-------
seqinfo: 93 sequences (1 circular) from hg19 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("chr1", 2),
             ranges=IRanges(start=c(100000,500000),
                            end = c(200000,600000)))

> transcriptsByOverlaps(txdb, gr)
GRanges object with 5 ranges and 2 metadata columns:
      seqnames            ranges strand |     tx_id     tx_name
         <Rle>         <IRanges>  <Rle> | <integer> <character>
  [1]     chr1 [568844, 568913]      + |        13  uc001abb.3
  [2]     chr1 [134773, 140566]      - |      4095  uc021oeg.2
  [3]     chr1 [566093, 566115]      - |      4096  uc021oej.1
  [4]     chr1 [566135, 566155]      - |      4097  uc021oek.1
  [5]     chr1 [566240, 566263]      - |      4098  uc021oel.1
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome
```

A practical example

- Assume I have a list of protein 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, a little slower.
- 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`.