Handling genomic data using Bioconductor I: Biostrings and BSgenome

Outline

- In next two classes, we will introduce functionalities of several Bioconductor packages for handling some genomic data:
  - DNA sequences.
  - Genomic intervals.
  - Genome annotations, e.g., genes, exons.

Motivating examples

- After “peak” (e.g., TFBS) detection from ChIP-seq:
  - locational distribution of binding sites, e.g., whether they are close to promoters, exons, introns, etc.
  - DNA sequence features (GC contents, CpG counts, etc.) of the binding sites.
  - motif enrichment of peaks.
- Comparative analyses:
  - overlaps of two lists of peaks.
  - relationships of TF binding and gene expressions.
- Obtaining read counts in specified genomic regions from second generation sequencing data.

These are routine works for a bioinformatician!

After these two classes

- You will be able to:
  - Quickly obtain sequences and genomic annotations for many species.
  - Explore the patterns for DNA sequences: sequence compositions, motif searches, etc.
  - Compare multiple lists of genomic intervals.

How to do these without Bioconductor

- For DNA sequence analysis:
  - Download the sequence file (fasta, a big plain text file) and write your own program to analyze it.
- For genome annotations:
  - Obtain the annotation file (like we did in lab 1 to download human genes) and analyze.
- Handling genomic intervals:
  - Some other software like BEDtools.

With Bioconductor

- Bioconductor provide many useful packages for efficiently handling genome data:
  - **`Biostrings`** defines containers and provides functions for genome sequence data.
  - **`BSgenome`** and other genome data packages provide full genome sequences for many species.
  - **`GenomicRanges`** handles genomic interval sets.
  - **`GenomicFeatures`** provide functions to retrieve and manage genomic features from public databases.

  These make our work (and life) a lot easier!

Biostrings

- Containers for representing (large) biological sequences.
- Provide a rich collection of utility functions for basic operations:
  - Storing, subsetting, matching and alignment.
- Computationally efficient:
  - Using bit pattern to encode the sequence data.

Operations on Single strings: *XString* class and subclasses

- *XString* is a "virtual class" and cannot be "instantiated" (cannot create a XString object).
- Four subclasses:
  - *BString*: for storing a general string.
  - *DNAString*: for storing a DNA (nucleotide) sequence.
  - *RNAString*: for storing a RNA sequence.
  - *AAString*: for storing protein (amino acid) sequence.
- Objects from all four subclasses operate similarly.

Basic operations of *BString*

- Create an object of Biostrings:

```
> library(Biostrings)
> a=BString("I am a string!")
> a
  14-letter "BString" instance
seq: I am a string!
> length(a)
[1] 14
```

- Subsetting:

```
> a[1:4]
  4-letter "BString" instance
seq: I am
> subseq(a,1,4)
  4-letter "BString" instance
seq: I am
```

The subseq function is more efficient than [] according to the manual.

- Revert

```
> rev(a)
  14-letter "BString" instance
seq: !gnirts a ma I
```

- Comparison and dump to a (real) string

```
> a=="I am"
[1] FALSE
> a[1:4]=="I am"
[1] TRUE
> toString(a)
[1] "I am a string!"
> class(a)
[1] "BString"
attr(,"package")
[1] "Biostrings"
> class(toString(a))
[1] "character"
```

DNAString/RNAString

Only difference is that they only take “valid” characters to represent nucleotides:

```
> IUPAC_CODE_MAP
     A      C      G      T      M      R      W      S      Y      K      V
   "A"    "C"    "G"    "T"   "AC"   "AG"   "AT"   "CG"   "CT"   "GT"  "ACG"
     H      D      B      N
 "ACT"  "AGT"  "CGT" "ACGT"
> DNA_ALPHABET
 [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+"
> DNA_BASES
[1] "A" "C" "G" "T"
> RNA_ALPHABET
 [1] "A" "C" "G" "U" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+"
> RNA_BASES
[1] "A" "C" "G" "U"
```

Creating DNA/RNA strings

```
> a=DNAString("I  am a string")
Error in .charToXString(basetype, x, start, end, width) :
 key 73 (char 'I') not in lookup table
> a=DNAString("ATTGCC")
> a
  6-letter "DNAString" instance
seq: ATTGCC
> b=RNAString("ATTGCC")
Error in .charToXString(basetype, x, start, end, width) :
 key 84 (char 'T') not in lookup table

> b=RNAString("AUUGCC")
> b
  6-letter "RNAString" instance
seq: AUUGCC
```

Simple frequency counting

```
> alphabetFrequency(a)
A C G T M R W S Y K V H D B N - +
1 2 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0

> alphabetFrequency(a,  baseOnly=TRUE)
    A     C     G     T other
    1     2     1     2     0

> letterFrequency(a,"C")
C
2

> letterFrequency(a,"CG")
C|G
  3
```

Complements

```
> a
  6-letter "DNAString" instance
seq: ATTGCC

> complement(a)
  6-letter "DNAString" instance
seq: TAACGG

> reverseComplement(a)
  6-letter "DNAString" instance
seq: GGCAAT
```

Single string matching and alignment

- Functions are divided into four groups:
  - Finding occurrences of a given pattern: *matchPattern, countPattern, vmatchPattern, vcountPattern*
  - Matching a dictionary of patterns against a reference: *matchPDict, countPDict*
  - Matching/counting with position Weight Matrix (PWM): *matchPWM, countPWM, PWMscoreStartingAt.*
  - Global/local alignment: *pairwiseAlignment, stringDist*

matchPattern

- Finds occurrences of a given pattern in a sequence, allowing mismatch and insertion/deletions (indels):

```
> a=DNAString("ACGTACGTACGC")
> matchPattern("CGT", a)
  Views on a 12-letter DNAString subject
subject: ACGTACGTACGC
views:
    start end width
[1]     2   4     3 [CGT]
[2]     6   8     3 [CGT]
> matchPattern("CGT", a, max.mismatch=1)
  Views on a 12-letter DNAString subject
subject: ACGTACGTACGC
views:
    start end width
[1]     2   4     3 [CGT]
[2]     6   8     3 [CGT]
[3]    10  12     3 [CGC]
```

```
> m=matchPattern("CGT", a, max.mismatch=1)
> start(m)
[1]  2  6 10
> end(m)
[1]  4  8 12
> length(m)
[1] 3
> countPattern("CGT", a, max.mismatch=1)
[1] 3
```

- These functions can be used to compute n-mer occurrence in a large genome efficiently. For example:
  - GC content: occurrence of “C” + occurrence of “G” (alternatively this can be obtained using frequency functions which is more efficient).
  - CpG content: occurrence of “CG”.

matchPDict

- Finding occurrence for a set of patterns.
  - Alternatively you can write a loop but this is much more efficient (R loops are notoriously slow).

```
> a=DNAString("ACGTACGTACGC")
> dict0=PDict(c("CGT","ACG"))
> mm=matchPDict(dict0, a)
> mm[[1]]
IRanges of length 2
    start end width
[1]     2   4     3
[2]     6   8     3
> mm[[2]]
IRanges of length 3
    start end width
[1]     1   3     3
[2]     5   7     3
[3]     9  11     3
```

Working with PWM

PWM: **Position Weight Matrix**, used to represent DNA motifs.

```
> a=DNAString("ACGTACGTACTC")
> motif=matrix(c(0.97,0.01,0.01,0.01,0.1,0.5,0.39,0.01,0.01,0.05,0.5,0.44),
    nrow=4)
> rownames(motif)=c("A","C","G","T")
> motif
  [,1] [,2] [,3]
A 0.97 0.10 0.01
C 0.01 0.50 0.05
G 0.01 0.39 0.50
T 0.01 0.01 0.44
> matchPWM(motif, a)
  Views on a 12-letter DNAString subject
subject: ACGTACGTACTC
views:
    start end width
[1]     1   3     3 [ACG]
[2]     5   7     3 [ACG]
[3]     9  11     3 [ACT]
> countPWM(motif, a)
[1] 3
> PWMscoreStartingAt(motif, a, 1:10)
 [1] 1.97 0.84 0.03 0.16 1.97 0.84 0.03 0.16 1.91 0.07
```

Operations on multiple strings: String views and set

- Operations on multiple strings can be achieved in a loop, but very inefficient.
- Multiple strings are derived from a “mother” string, and put into a string “view” or a “set”.
- *XStringViews:* contains multiple “views” (start/end locations) of the same string.
- DNAStringSet/RNAStringSet: similar but created actual DNA/RNAString instances.
- StringSet allows more operations than StringViews.

Basic operations on *XStringViews*

```
> a=DNAString("ACGTACGTACTC")
> a2=Views(a, start=c(1,5,8), end=c(3,8,12))
> a2
  Views on a 12-letter DNAString subject
subject: ACGTACGTACTC
views:
    start end width
[1]     1   3     3 [ACG]
[2]     5   8     4 [ACGT]
[3]     8  12     5 [TACTC]
> subject(a2)
  12-letter "DNAString" instance
seq: ACGTACGTACTC
> length(a2)
[1] 3
> start(a2)
[1] 1 5 8
> end(a2)
[1]  3  8 12
```

```
> alphabetFrequency(a2, baseOnly=TRUE)
     A C G T other
[1,] 1 1 1 0     0
[2,] 1 1 1 1     0
[3,] 1 2 0 2     0

> a2==DNAString("ACGT")
[1]  TRUE  TRUE FALSE

> toString(a2)
[1] "ACGT, ACGT, ACTC”
```

Basic operations on *DNAStringSet*

```
> a=DNAString("ACGTACGTACTC")
> a2=DNAStringSet(a, start=c(1,5,9), end=c(4,8,12))
> a2
  A DNAStringSet instance of length 3
    width seq
[1]     4 ACGT
[2]     4 ACGT
[3]     4 ACTC
> a2[[1]]
  4-letter "DNAString" instance
seq: ACGT
> alphabetFrequency(a2, baseOnly=TRUE)
     A C G T other
[1,] 1 1 1 1     0
[2,] 1 1 1 1     0
[3,] 1 2 0 1     0
```

Some Operations only allowed for StringSet not Views, such as set operations

```
> a1=DNAStringSet(a, start=c(1,5,9), end=c(4,8,12))
> a1
  A DNAStringSet instance of length 3
    width seq
[1]     4 ACGT
[2]     4 ACGT
[3]     4 ACTC
> unique(a1)
  A DNAStringSet instance of length 2
    width seq
[1]     4 ACGT
[2]     4 ACTC
> a2=Views(a, start=c(1,5,9), end=c(4,8,12))
> unique(a2)
Error in duplicated.default(x, incomparables = incomparables, ...) :
  duplicated() applies only to vectors
```

```
> a1=Views(a, start=c(1,9), end=c(4,12))
> a1
  Views on a 12-letter DNAString subject
subject: ACGTACGTACTC
views:
    start end width
[1]     1   4     4 [ACGT]
[2]     9  12     4 [ACTC]
> a2=Views(a, start=c(1), end=c(4))
> a2
  Views on a 12-letter DNAString subject
subject: ACGTACGTACTC
views:
    start end width
[1]     1   4     4 [ACGT]
> setdiff(a1,a2) ## this will generate error
Error in as.vector(x) : no method for coercing this S4 class to a vector
> union(a1, a2)
Error in as.vector(x) : no method for coercing this S4 class to a vector
```

```
> a1=DNAStringSet(a, start=c(1,9), end=c(4,12))
> a1
  A DNAStringSet instance of length 2
    width seq
[1]     4 ACGT
[2]     4 ACTC
> a2=DNAStringSet(a, start=c(1), end=c(4))
> a2
  A DNAStringSet instance of length 1
    width seq
[1]     4 ACGT
> setdiff(a1,a2)
  A DNAStringSet instance of length 1
    width seq
[1]     4 ACTC
> union(a1,a2)
  A DNAStringSet instance of length 2
    width seq
[1]     4 ACGT
[2]     4 ACTC
```

Matching with multiple strings

- Use vmatchPattern and vmatchPDict.
- No corresponding function for PWM.

```
> a=DNAString("ACGTACGTACTC")
> a2=DNAStringSet(a, start=c(1,5,9), end=c(4,8,12))
> vv=vmatchPattern("CG", a2)
> vv
MIndex object of length 3
> vv[[1]]
IRanges of length 1
    start end width
[1]     2   3     2
```

These don’t work for Views

```
> a2=Views(a, start=c(1,5,9), end=c(4,8,12))
> a2
  Views on a 12-letter DNAString subject
subject: ACGTACGTACTC
views:
    start end width
[1]     1   4     4 [ACGT]
[2]     5   8     4 [ACGT]
[3]     9  12     4 [ACTC]

> vv=vmatchPattern("CG", a2)
Error in .local(pattern, subject, max.mismatch, min.mismatch,
with.indels,  :
  XStringViews objects are not supported yet, sorry
```

`BSgenome` and genome data packages

- BSgenome: provides the infrastructure and higher level functions.
- Genome data packages:
  - Provide whole genome sequences for many genomes (77 so far).
  - Naming rule: `BSgenome.species.provider.build`.
  - Data stored in basic containers defined in Biostrings, e.g., DNAString.
  - Need to be installed individually using `biocLite()`, Like other bioconductor packages.

Available genomes data

```
> available.genomes()
 [1] "BSgenome.Alyrata.JGI.v1"
 [2] "BSgenome.Amellifera.BeeBase.assembly4"
 [3] "BSgenome.Amellifera.UCSC.apiMel2"
 [4] "BSgenome.Amellifera.UCSC.apiMel2.masked"
 [5] "BSgenome.Athaliana.TAIR.04232008"
 [6] "BSgenome.Athaliana.TAIR.TAIR9"
 [7] "BSgenome.Btaurus.UCSC.bosTau3"
 [8] "BSgenome.Btaurus.UCSC.bosTau3.masked"
 [9] "BSgenome.Btaurus.UCSC.bosTau4"
[10] "BSgenome.Btaurus.UCSC.bosTau4.masked"
...
[79] "BSgenome.Tguttata.UCSC.taeGut1”
[80] "BSgenome.Tguttata.UCSC.taeGut1.masked"
[81] "BSgenome.Tguttata.UCSC.taeGut2"
[82] "BSgenome.Vvinifera.URGI.IGGP12Xv0"
[83] "BSgenome.Vvinifera.URGI.IGGP12Xv2"
[84] "BSgenome.Vvinifera.URGI.IGGP8X"
```

Load the genome data package

```
> library(BSgenome.Hsapiens.UCSC.hg18)
> ls("package:BSgenome.Hsapiens.UCSC.hg18")
[1] "BSgenome.Hsapiens.UCSC.hg18"  "Hsapiens"
> Hsapiens
Human genome:
# organism: Homo sapiens (Human)
# provider: UCSC# provider version: hg18
# release date: Mar. 2006# release name: NCBI Build 36.1
# 49 sequences:
#   chr1           chr2           chr3           chr4           chr5
#   chr6           chr7           chr8           chr9           chr10
#   chr11          chr12          chr13          chr14          chr15
#   chr16          chr17          chr18          chr19          chr20
#   chr21          chr22          chrX           chrY           chrM
#   chr5_h2_hap1   chr6_cox_hap1  chr6_qbl_hap2  chr22_h2_hap1  chr1_random
#   chr2_random    chr3_random    chr4_random    chr5_random    chr6_random
#   chr7_random    chr8_random    chr9_random    chr10_random   chr11_random
#   chr13_random   chr15_random   chr16_random   chr17_random   chr18_random
#   chr19_random   chr21_random   chr22_random   chrX_random
# (use 'seqnames()' to see all the sequence names, use the '$' or '[[' operator
# to access a given sequence)
```

Basic operations

- Access the sequence:

```
> Hsapiens$chr1

247249719-letter "DNAString" instance

seq: TAACCCTAACCCTAACCCTAACCCTAACCCTAACCC...NNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

- Data are not loaded until accessed.
- Some simple information can be obtained without loading in the data:

```
> seqnames(Hsapiens)
 [1] "chr1"          "chr2"          "chr3"          "chr4"
 [5] "chr5"          "chr6"          "chr7"          "chr8"
 [9] "chr9"          "chr10"         "chr11"         "chr12"
[13] "chr13"         "chr14"         "chr15"         "chr16"
[17] "chr17"         "chr18"         "chr19"         "chr20"
[21] "chr21"         "chr22"         "chrX"          "chrY"
...

> seqlengths(Hsapiens)
      chr1       chr2       chr3       chr4       chr5
 247249719  242951149  199501827  191273063  180857866
      chr6       chr7       chr8       chr9      chr10
 170899992  158821424  146274826  140273252  135374737
     chr11      chr12      chr13      chr14      chr15
 134452384  132349534  114142980  106368585  100338915
     chr16      chr17      chr18      chr19      chr20
  88827254   78774742   76117153   63811651   62435964
     chr21      chr22       chrX       chrY       chrM
  46944323   49691432  154913754   57772954      16571
 ...
```

Counting and matching

```
> alphabetFrequency(Hsapiens$chr1, baseOnly=TRUE)
       A        C        G        T    other
65491918 46964756 46956489 65586556 22250000

> alphabetFrequency(Hsapiens$chr1, baseOnly=TRUE) / length(Hsapiens$chr1)
         A          C          G          T      other
0.26488167 0.18994867 0.18991524 0.26526443 0.08998999

> mm=matchPattern("CG", Hsapiens$chr1)
> length(mm)
[1] 2281713
```

```
> mm
  Views on a 247249719-letter DNAString subject
subject:
TAACCCTAACCCTAACCCTAACCCTAACCCTAAC...NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
views:
               start       end width
      [1]        469       470     2 [CG]
      [2]        471       472     2 [CG]
      [3]        484       485     2 [CG]
      [4]        489       490     2 [CG]
      [5]        493       494     2 [CG]
      ...        ...       ...   ... ...
[2281709] 247199441 247199442     2 [CG]
[2281710] 247199447 247199448     2 [CG]
[2281711] 247199491 247199492     2 [CG]
[2281712] 247199632 247199633     2 [CG]
[2281713] 247199679 247199680     2 [CG]
```

SNPs

- SNP information from dbSNP are available:

```
> available.SNPs()
[1] "SNPlocs.Hsapiens.dbSNP.20090506" "SNPlocs.Hsapiens.dbSNP.20100427"
[3] "SNPlocs.Hsapiens.dbSNP.20101109" "SNPlocs.Hsapiens.dbSNP.20110815"
[5] "SNPlocs.Hsapiens.dbSNP.20111119” "SNPlocs.Hsapiens.dbSNP.20120608"
```

- SNP data can be installed as other Bioconductor packages:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("SNPlocs.Hsapiens.dbSNP.20120608”)

> installed.SNPs()
[1] "SNPlocs.Hsapiens.dbSNP.20120608"
```

SNP injection

- SNPs can be “injected” into the reference genome to create a reference with the SNPs.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> SnpHsapiens <- injectSNPs(Hsapiens,
    "SNPlocs.Hsapiens.dbSNP.20120608")
> SnpHsapiens
Human genome
|
| organism: Homo sapiens (Human)
| provider: UCSC
| provider version: hg19
| release date: Feb. 2009
| release name: Genome Reference Consortium GRCh37
| with SNPs injected from package:SNPlocs.Hsapiens.dbSNP.20120608
```

More on SNPs

```
> snpcount(SnpHsapiens)
chr1    chr2    chr3    chr4    chr5    chr6    chr7    chr8    chr9   chr10
3517088 3751808 3130073 3091112 2827921 2752803 2557048 2455150 1944002 2168513
  chr11   chr12   chr13   chr14   chr15   chr16   chr17   chr18   chr19   chr20
2196807 2104180 1534404 1421772 1299703 1452916 1247508 1224385  997260 1052332
  chr21   chr22    chrX    chrY
 632005  609352 1644601   84028

> head(snplocs(SnpHsapiens, "chr1"))

  RefSNP_id alleles_as_ambig   loc
1 201752861                M 10177
2 201694901                Y 10180
3 200279319                M 10231
4 145599635                Y 10234
5 148908337                W 10248
6 199706086                M 10250

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

Review

- We have introduced following useful Bioconductor package: **`Biostrings,BSgenome`**.
- To do after this class:
  - Install following Bioconductor packages on your computer: `Biostrings, BSgenome, BSgenome.Celegans.UCSC.ce2, BSgenome.Hsapiens.UCSC.hg19.`
  - Review slides and rerun the R codes (on the class webpage).
  - Start to think about final project topic.