An introduction to the Biostrings/BSgenome framework

Hervé Pagès and Patrick Aboyoun
Computational Biology Program
Fred Hutchinson Cancer Research Center

Biostrings

- Containers for representing large biological sequences (DNA/RNA/amino acids)
- Utilities for basic computations on sequences
- Tools for sequence matching and pairwise alignments

BSgenome data packages

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

Biostrings
Basic containers

- Single sequence: `XString` (virtual class) and its direct extensions `BString`, `DNAString`, `RNAString` and `AAString`
- Set of sequences: `XStringSet` (virtual class) and its direct extensions `BStringSet`, `DNAStringSet`, `RNAStringSet` and `AAStringSet`
- Set of views on a sequence: `XStringViews`
- Masked sequence: Masked`XString` (virtual class) and its direct extensions Masked`BString`, Masked`DNAString`, Masked`RNAString` and Masked`AAString`

Biostrings
Utilities

Extract a subsequence: `subseq()`

```
> library(BSgenome.Celegans.UCSC.ce2)
> chrI <- Celegans$chrI
> chrI
  15080483-letter "DNAString" instance
seq: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCC...GGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
> subseq(chrI, start=1000, end=-1000)
  15078485-letter "DNAString" instance
seq: ATTTTTCGGGTTTTTTGAAATGAATATCGTAGC...TTTAAACTCGTATCGGTTAACCAACCCTTGGAT
```

IO: `read.DNAStringSet()`, `write.XStringSet()`

```
> orfs <- read.DNAStringSet(file, "fasta")
Read 450 items
> orfs
  A DNAStringSet instance of length 7
    width seq                                               names
[1]  5573 ACTTGTAAATATATCTTTTA...ATCGACCTTATTGTTGATAT YAL001C TFC3 SGDI...
[2]  5825 TTCCAAGGCCGATGAATTCG...AAATTTTTTTCTATTCTCTT YAL002W VPS8 SGDI...
[3]  2987 CTTCATGTCAGCCTGCACTT...TACTCATGTAGCTGCCTCAT YAL003W EFB1 SGDI...
[4]  3929 CACTCATATCGGGGGTCTTA...CCCGAAACACGAAAAAGTAC YAL005C SSA1 SGDI...
[5]  2648 AGAGAAAGAGTTTCACTTCT...TAATTTATGTGTGAACATAG YAL007C ERP2 SGDI...
[6]  2597 GTGTCCGGGCCTCGCAGGCG...TTTTGGCAGAATGTACTTTT YAL008W FUN14 SGD...
[7]  2780 CAAGATAATGTCAAAGTTAG...AAGGAAGAAAAAAAAATCAC YAL009W SPO7 SGDI...
```

Biostrings
Utilities

Coerce an XStringSet object to another type of XStringSet or/and trim its elements with `BStringSet(), DNAStringSet(), etc...`

```
> RNAStringSet(orfs)
  A RNAStringSet instance of length 7
    width seq                                               names
[1]  5573 ACUUGUAAAUAUAUCUUUUA...AUCGACCUUAUUGUUGAUAU YAL001C TFC3 SGDI...
[2]  5825 UUCCAAGGCCGAUGAAUUCG...AAAUUUUUUUCUAUUCUCUU YAL002W VPS8 SGDI...
[3]  2987 CUUCAUGUCAGCCUGCACUU...UACUCAUGUAGCUGCCUCAU YAL003W EFB1 SGDI...
[4]  3929 CACUCAUAUCGGGGGUCUUA...CCCGAAACACGAAAAAGUAC YAL005C SSA1 SGDI...
[5]  2648 AGAGAAAGAGUUUCACUUCU...UAAUUUAUGUGUGAACAUAG YAL007C ERP2 SGDI...
[6]  2597 GUGUCCGGGCCUCGCAGGCG...UUUUGGCAGAAUGUACUUUU YAL008W FUN14 SGD...
[7]  2780 CAAGAUAAUGUCAAAGUUAG...AAGGAAGAAAAAAAAAAUCAC YAL009W SPO7 SGDI...
> RNAStringSet(orfs, end=12)
  A RNAStringSet instance of length 7
    width seq                                               names
[1]    12 ACUUGUAAAUAU                                      YAL001C TFC3 SGDI...
[2]    12 UUCCAAGGCCGA                                      YAL002W VPS8 SGDI...
[3]    12 CUUCAUGUCAGC                                      YAL003W EFB1 SGDI...
[4]    12 CACUCAUAUCGG                                      YAL005C SSA1 SGDI...
[5]    12 AGAGAAAGAGUU                                      YAL007C ERP2 SGDI...
[6]    12 GUGUCCGGGCCU                                      YAL008W FUN14 SGD...
[7]    12 CAAGAUAAUGUC                                      YAL009W SPO7 SGDI...
> AAStringSet(orfs)
Error in getXStringSubtypeConversionLookup(from_baseClass, baseClass) :
  incompatible XString/XStringSet subtypes
```

Biostrings
Utilities

Basic transformations: `reverse()`, `complement()`, `reverseComplement()`, `translate()`

```
> x
  21-letter "DNAString" instance
seq: TCAACGTTGAATAGCGTACCG
> reverseComplement(x)
  21-letter "DNAString" instance
seq: CGGTACGCTATTCAACGTTGA
> translate(x)
  7-letter "AAString" instance
seq: STLNSVP
> translate(reverseComplement(x))
  7-letter "AAString" instance
seq: RYAIQR*
```

Character translation: `chartr()`

```
> library(BSgenome.Celegans.UCSC.ce2)
> chrII <- Celegans$chrII
> alphabetFrequency(chrII, baseOnly=TRUE)
      A       C       G       T   other
4878194 2769208 2762193 4869710       3
> chrIIbis <- chartr("C", "T", chrII)
> chrIIbis
  15279308-letter "DNAString" instance
seq: TTTAAGTTTAAGTTTAAGTTTAAGTTTAAGTTT...AGGTTGAGATTTAGGTTTAGGTTTAGGTTTAGT
> alphabetFrequency(chrIIbis, baseOnly=TRUE)
      A       C       G       T   other
4878194       0 2762193 7638918       3
```

Biostrings
Utilities

alphabetFrequency(), uniqueLetters()

```
> yeast1
  230208-letter "DNAString" instance
seq: CCACACCACACCCACACACCCACACACCACACC...GTGTGGGTGTGGTGTGGGTGTGGTGTGTGTGGG
> alphabetFrequency(yeast1)
    A     C     G     T     M     R     W     S     Y     K     V     H
69830 44643 45765 69970     0     0     0     0     0     0     0     0
    D     B     N     -     +
    0     0     0     0     0
> alphabetFrequency(yeast1, baseOnly=TRUE)
    A     C     G     T other
69830 44643 45765 69970     0
> uniqueLetters(yeast1)
[1] "A" "C" "G" "T"
```

dinucleotideFrequency(), trinucleotideFrequency(), oligonucleotideFrequency()

```
> dinucleotideFrequency(yeast1)
   AA    AC    AG    AT    CA    CC    CG    CT    GA    GC    GG    GT
23947 12493 13621 19769 15224  9218  7089 13112 14478  8910  9438 12938
   TA    TC    TG    TT
16181 14021 15617 24151
> head(trinucleotideFrequency(yeast1))
 AAA  AAC  AAG  AAT  ACA  ACC
8576 4105 4960 6306 3924 2849
```

Biostrings
String matching

- A common problem: find all the occurences (aka *matches* or *hits*) of a given *pattern* (typically short) in a (typically long) reference sequence (aka the *subject*)

pattern: ATGAT

subject:

```
...ACGAGATTTATGATGATCGGATTATACGACACCGATCGGCCATATGATTAC...
               ---   --  ---                  -----
```

- *Exact* match = the sequence in the match is identical to the pattern
- *Inexact* match = some differences are allowed
  - Allow a maximum number of mismatching letters per match (max.mismatch argument)
  - Allow indels i.e. the “edit distance” between a match and the pattern must be < arbitrary value
  - IUPAC ambiguity letters in the pattern: interpret them litterally or allow them to match the base letters they stand for?

Biostrings
String matching

- **`matchPattern()`**: **one** pattern, **one** reference sequence in the subject

  pattern: `ATGAT`

  subject: `TTTACGAGATTTATGATGATCGGATTATACAAG`

- **`vmatchPattern()`**: **one** pattern, **many** reference sequences in the subject

  pattern: `ATGAT`

  subject:

```
ACGGAATTAGACCAT
TTTACGAGATTTATGATGATCGGATTATACAAG
...
TGGACAGGTACGTAGGATGCGGTTA
```

Biostrings
String matching

- **`matchPDict()`**: **many** patterns, **one** reference sequence in the subject

  patterns:

  ```
  ATGAT
  AGTTC
  TTCAC
  TGCTA
  ...
  GATGC
  ```

  subject:

  ```
  TTTACGAGATTTATGATGATCGGATTATACAAG
  ```

  - The set of patterns (aka the *dictionary*) needs to be prepreocessed first (stored in a PDict object)
  - Preprocessing is fast but uses a lot of memory (e.g. < 40 sec. and 2-3 GB for 5-10 millions 36-mers)
  - Some of the search parameters (e.g. exact/inexact matching) must be decided at preprocessing time (by defining a Trusted Band)

- [In the TODO pipe] **`vmatchPDict()`**: **many** patterns, **many** reference sequences in the subject

  patterns:

  ```
  ATGAT
  AGTTC
  TTCAC
  TGCTA
  ...
  GATGC
  ```

  subject:

  ```
  ACGGAATTAGACCAT
  TTTACGAGATTTATGATGATCGGATTATACAAG
  ...
  TGGACAGGTACGTAGGATGCGGTTA
  ```

Biostrings
String matching

- EXAMPLE 1: Using `matchPDict()` for **exact** matching of a **constant-width** dictionary

1. Load the dictionary

```
> library(hgu95av2probe)
> dict0 <- DNAStringSet(hgu95av2probe)
> dict0
  A DNAStringSet instance of length 201800
         width seq
     [1]    25 TGGCTCCTGCTGAGGTCCCCTTTCC
     [2]    25 GGCTGTGAATTCCTGTACATATTTC
     [3]    25 GCTTCAATTCCATTATGTTTTAATG
     [4]    25 GCCGTTTGACAGAGCATGCTCTGCG
     [5]    25 TGACAGAGCATGCTCTGCGTTGTTG
     [6]    25 CTCTGCGTTGTTGGTTTCACCAGCT
     [7]    25 GGTTTCACCAGCTTCTGCCCTCACA
     [8]    25 TTCTGCCCTCACATGCACAGGGATT
     [9]    25 CCTCACATGCACAGGGATTTAACAA
     ...   ... ...
[201792]    25 GAGTGCCAATTCGATGATGAGTCAG
[201793]    25 ACACTGACACTTGTGCTCCTTGTCA
[201794]    25 CAATTCGATGATGAGTCAGCAACTG
[201795]    25 GACTTTCTGAGGAGATGGATAGCCT
[201796]    25 AGATGGATAGCCTTCTGTCAAAGCA
[201797]    25 ATAGCCTTCTGTCAAAGCATCATCT
[201798]    25 TTCTGTCAAAGCATCATCTCAACAA
[201799]    25 CAAAGCATCATCTCAACAAGCCCTC
[201800]    25 GTGCTCCTTGTCAACAGCGCACCCA
```

Biostrings
String matching

- EXAMPLE 1: Using `matchPDict()` for **exact** matching of a **constant-width** dictionary

2. Preprocess the dictionary

```
> pdict <- PDict(dict0)  # takes < 5 sec.
```

3. Load the subject

```
> library(BSgenome.Hsapiens.UCSC.hg18)
> chr1 <- unmasked(Hsapiens$chr1)
> chr1
  247249719-letter "DNAString" instance
seq:
TAACCCTAACCCTAACCCTAACCCTAACCC...NNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

4. Call matchPDict()

```
> m <- matchPDict(pdict, chr1)  # takes < 30 sec.
> m
201800-pattern MIndex object
```

5. Query the MIndex object

```
> m[[700]]  # extract hits for pattern 700
IRanges object:
      start       end width
1  58787625  58787649    25
2 110757441 110757465    25
3 195332894 195332918    25
```

Biostrings
String matching

Calling `PDict(dict0)` with **no additional argument** (like in EXAMPLE 1) only works on a **constant-width** dictionary and returns a PDict object that can only be used for **exact** matching!

- EXAMPLE 2: Using `matchPDict()` with an explicit **Trusted Band** defined on the dictionary

Trusted Band

```
ATGAT TCTAGTGACTGTGA AATGAT
AGTTC AATGATTATAGTAC CAGATGATT
TTCAC CTGTCACCATGTAG CACGAGG
TGCTA ATGGTATTAATTAT TTAACAAACTA
...
GATGC ATGTATTAGTGATA GGAGCATTAG
TCCGG CGCCTGAGCTAGGA CATTAAAC
```

head tail

- Matching must be **exact** on the Trusted Band
- Matching can be **inexact** on the *head* and *tail* of the dictionary (how inexact will be decided later when calling `matchPDict()`)
- The dictionary doesn't have to be of constant width anymore
- Only Trusted Bands of constant widths are supported

Preprocess with

```
> pdict <- PDict(dict0, tb.start=6, tb.end=19)
```

Biostrings
String matching

- EXAMPLE 2: Using `matchPDict()` with an explicit **Trusted Band** defined on the dictionary

Call matchPDict()

```
> pdict
TB_PDict object of length 201800, width 25, and type "ACtree":
  - with a head of width 5
  - with a Trusted Band of width 14
  - with a tail of width 6
> m <- matchPDict(pdict, chr1, max.mismatch=3)
> m
201800-pattern MIndex object
```

Query the MIndex object

```
> endIndex(m)[[700]]
[1]  24302718  58787649 110757465 187194793 195332918
```

A two-stage algorithm is used:

(1) Find all the exact matches for all the trimmed patterns that are in the Trusted Band -> this gives one set of provisory matches per pattern

(2) For each pattern and each of its provisory match: compare the head and tail of the pattern with the flanking sequences of the match and reject or keep the match

Tip: a narrow Trusted Band doesn't perform well. Try to keep its width >= 12.

Biostrings
String matching

- EXAMPLE 3: Allow a **small** number of mismatches **anywhere** in the patterns of a **constant-width** dictionary with `PDict()/matchPDict()`

Preprocess the dictionary

```
> pdict <- PDict(dict0, max.mismatch=1)
> pdict
  MTB_PDict object of length 201800 and width 25
Components:
[[1]]
TB_PDict object of length 201800, width 25, and type "ACtree":
  - with NO head
  - with a Trusted Band of width 12
  - with a tail of width 13

[[2]]
TB_PDict object of length 201800, width 25, and type "ACtree":
  - with a head of width 12
  - with a Trusted Band of width 13
  - with NO tail
```

Biostrings
String matching

- EXAMPLE 3: Allow a **small** number of mismatches **anywhere** in the patterns of a **constant-width** dictionary with `PDict()/matchPDict()`

Call matchPDict()

```
> m <- matchPDict(pdict, chr1, max.mismatch=1)  # takes < 2 min.
> m
201800-pattern MIndex object
```

Query the MIndex object

```
> endIndex(m)[[700]]
[1]  24302718  58787649 110757465 195332918
```

How it works:

- For each implicit Trusted Band (2 in our example) the two-stage algorithm described previously is used => This produces 2 sets of matches per pattern.
- For each pattern the 2 sets of matches are combined in a single set (duplicates are removed and matches are ordered by ascending start positions)
- Performance will degrade fast with this dictionary if we try to use max.mismatch = 2 or 3 because the implicit Trusted Bands will become too narrow (only 8 nucleotides if max.mismatch = 2 => average of 4000 provisory matches per pattern found in chr1 during stage 1).

Biostrings
String matching

- **`countPattern()`**, **`vcountPattern()`**, **`countPDict()`**, **`vcountPDict()`**: like the *`match*()`* functions but return the match count only (much less memory needed)

Other tools for sequence matching and alignments:

- **`matchLRPatterns()`**: matches specified by a left pattern, a right pattern and a maximum distance them
- **`matchProbePair()`**: find amplicons given by a pair of primers (simulate PCR)
- **`MatchPWM()`**, **`matchWCP()`**: find motifs described by a Position Weight Matrix (PWM) or by Weighted Clustered Positions
- **`findPalindromes()`** / **`findComplementedPalindromes()`**
- **`pairwiseAlignment()`**: solves (Needleman-Wunsch) global alignment, (Smith-Waterman) local alignment, and (ends-free) overlap alignment problems

Support indels:

- `pairwiseAlignment()`, `matchPattern()/countPattern()`, `vcountPattern()`
- In the TODO pipe: `vmatchPattern()`, `matchPDict()`, `countPDict()`, ...

BSgenome data packages

- Full genome sequences stored in Biostrings containers / 1 genome per package
- 13 organisms / 21 packages in the current release (BioC 2.5)
- Most (but not all) packages contain sequences with built-in masks
- Naming convention: `BSgenome.`*Organism*`.`*Provider*`.`*BuildVersion*
- Use **`available.genomes()`** (from the BSgenome software package) to get the list:

```
> library(BSgenome)
> available.genomes()
 [1] "BSgenome.Amellifera.BeeBase.assembly4"
 [2] "BSgenome.Amellifera.UCSC.apiMel2"
 [3] "BSgenome.Athaliana.TAIR.01222004"
 [4] "BSgenome.Athaliana.TAIR.04232008"
 [5] "BSgenome.Btaurus.UCSC.bosTau3"
 [6] "BSgenome.Btaurus.UCSC.bosTau4"
 [7] "BSgenome.Celegans.UCSC.ce2"
 [8] "BSgenome.Cfamiliaris.UCSC.canFam2"
 [9] "BSgenome.Dmelanogaster.UCSC.dm2"
[10] "BSgenome.Dmelanogaster.UCSC.dm3"
[11] "BSgenome.Drerio.UCSC.danRer5"
[12] "BSgenome.Ecoli.NCBI.20080805"
[13] "BSgenome.Ggallus.UCSC.galGal3"
[14] "BSgenome.Hsapiens.UCSC.hg17"
[15] "BSgenome.Hsapiens.UCSC.hg18"
[16] "BSgenome.Hsapiens.UCSC.hg19"
[17] "BSgenome.Mmusculus.UCSC.mm8"
[18] "BSgenome.Mmusculus.UCSC.mm9"
[19] "BSgenome.Ptroglodytes.UCSC.panTro2"
[20] "BSgenome.Rnorvegicus.UCSC.rn4"
[21] "BSgenome.Scerevisiae.UCSC.sacCer1"
```

BSgenome data packages

- A BSgenome data package with no built-in masks:

```
> library(BSgenome.Celegans.UCSC.ce2)
> Celegans
Worm genome
|
| organism: Caenorhabditis elegans
| provider: UCSC
| provider version: ce2
| release date: Mar. 2004
| release name: WormBase v. WS120
|
| single sequences (see '?seqnames'):
|   chrI    chrII   chrIII  chrIV   chrV    chrX    chrM
|
| multiple sequences (see '?mseqnames'):
|   upstream1000  upstream2000  upstream5000
|
| (use the '$' or '[[' operator to access a given sequence)
> Celegans$chrI
  15080483-letter "DNAString" instance
seq: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTAA...TTAGGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
> class(Celegans$chrI)
[1] "DNAString"
attr(,"package")
[1] "Biostrings"
```

BSgenome data packages

- **`seqnames()`**, **`seqlengths()`** (do not load the sequences)

```
> seqnames(Celegans)
[1] "chrI"   "chrII"  "chrIII" "chrIV"  "chrV"   "chrX"   "chrM"
> seqlengths(Celegans)
    chrI    chrII   chrIII    chrIV     chrV     chrX     chrM
15080483 15279308 13783313 17493791 20922231 17718849    13794
```

- Use standard `lapply()`/`sapply()` to apply the same function to all the chromosomes

```
> sapply(seqnames(Celegans),
    function(seqname)
      alphabetFrequency(Celegans[[seqname]], baseOnly=TRUE))
         chrI   chrII  chrIII   chrIV    chrV    chrX chrM
A     4838561 4878194 4444527 5711041 6749806 5746418 4335
C     2697177 2769208 2449074 3034771 3711722 3119282 1225
G     2693544 2762193 2466260 3017009 3700959 3118284 2055
T     4851201 4869710 4423447 5730970 6759744 5734865 6179
other       0       3       5       0       0       0    0
```

  - Here all the sequences were loaded
  - Applying `colSums()` to this matrix would give the same result as `seqlengths(Celegans)`

BSgenome data packages

- A BSgenome data package with built-in masks:

```
> library(BSgenome.Btaurus.UCSC.bosTau4)
> Btaurus
Cow genome
|
| organism: Bos taurus
| provider: UCSC
| provider version: bosTau4
| release date: Oct. 2007
| release name: Baylor College of Medicine HGSC Btau_4.0
|
| single sequences (see '?seqnames'):
|   chr1   chr2   chr3   chr4   chr5   chr6   chr7   chr8   chr9   chr10  chr11
|   chr12  chr13  chr14  chr15  chr16  chr17  chr18  chr19  chr20  chr21  chr22
|   chr23  chr24  chr25  chr26  chr27  chr28  chr29  chrX   chrM
|
| multiple sequences (see '?mseqnames'):
|   chrUn.scaffolds  upstream1000     upstream2000     upstream5000
|
| (use the '$' or '[[' operator to access a given sequence)
```

BSgenome data packages

- A BSgenome data package with built-in masks:

```
> Btaurus$chr1
  161106243-letter "MaskedDNAString" instance (# for masking)
seq: TACCCCACTCACACTTATGGATAGATCAACTAAACA...TCCATTTTAGTTTATTTTTTTGTATGGTAGAATACT
masks:
  maskedwidth  maskedratio active names                                desc
1    11225171 6.967558e-02   TRUE AGAPS                      assembly gaps
2        1098 6.815378e-06   TRUE   AMB           intra-contig ambiguities
3    72217189 4.482582e-01  FALSE    RM                        RepeatMasker
4      314874 1.954449e-03  FALSE   TRF Tandem Repeats Finder [period<=12]
all masks together:
  maskedwidth maskedratio
     83443591   0.5179414
all active masks together:
  maskedwidth maskedratio
     11226269   0.0696824
> class(Btaurus$chr1)
[1] "MaskedDNAString"
attr(,"package")
[1] "Biostrings"
> masknames(Btaurus)
[1] "AGAPS" "AMB"   "RM"    "TRF"
```

- A mask can be **active** (the masked regions will be skipped during most computations) or **inactive** (the mask will be ignored). The user can toggle this.
- The set of built-in masks is guaranteed to be the same for all the single sequences in a given package (same order and same active masks too)

BSgenome data packages

Built-in mask names

(1) AGAPS: mask of assembly gaps

(2) AMB: mask of intra-contig ambiguities

(3) RM: mask of repeat regions as determined by the RepeatMasker software

(4) TRF: mask of repeat regions as determined by the Tandem Repeats Finder software (where only repeats with period less than or equal to 12 were kept)

Note that, in theory, masks 2, 3 and 4 should never overlap with mask 1.

```
> unmasked(chr1)
  161106243-letter "DNAString" instance
seq: TACCCCACTCACACTTATGGATAGATCAACTAAACA...TCCATTTTAGTTTATTTTTTTGTATGGTAGAATACT
> uniqueLetters(unmasked(chr1))
[1] "A" "C" "G" "T" "N"
> uniqueLetters(chr1)
[1] "A" "C" "G" "T"
> uniqueLetters(gaps(chr1))
[1] "N"
> active(masks(chr1))["AMB"] <- FALSE
> uniqueLetters(chr1)
[1] "A" "C" "G" "T" "N"
```