

A Tour of the IRanges/Biostrings/BSgenome Framework

Patrick Aboyoun
Computational Biology
Fred Hutchinson Cancer Research Center

IRanges

- Underlying infrastructure to manage and manipulate large sequences and “views” of their subsequences
- Emerging infrastructure for representing and leveraging annotations on sequence regions

Biostrings

- Builds on IRanges infrastructure to represent large biological character sequences (DNA/RNA/amino acids)
- Utilities for basic computations on these character sequences
- Tools for sequence matching and pairwise alignment

BSgenome data packages

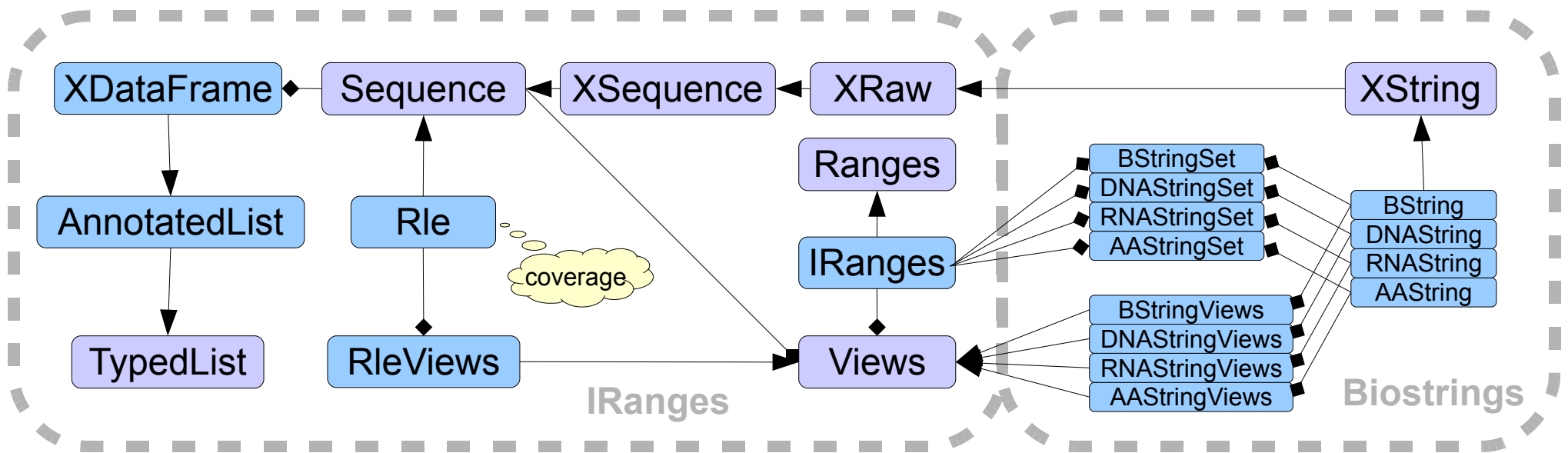
- Full genomes stored in Biostrings containers
- Currently 19 organisms supported (Human, Mouse, Worm, Yeast, etc.)
- Facilities for supporting new genomes (BSgenomeForge)

IRanges

- Package name derived from it's sequence index range class

```
> r <- IRanges(start=seq.int(from=400, by=-50, length=4), width=c(20, 18))
> r
IRanges object:
  start end width
1   400  419   20
2   350  367   18
3   300  319   20
4   250  267   18
```

- Forms basis of sequence data infrastructure



IRanges

- Two main types of sequences: Rle objects (IRanges) and XString objects (Biostrings)
- Run-length encoded sequences are ideal for sequences with repeated values (e.g. coverage)

```
> set.seed(123)
> x <- Rle(values=sample(0:8, 10000, replace=TRUE, prob=c(0.64, (8:1)/100)),
+         lengths=sample(1:200, 10000, replace=TRUE))
> x
'integer' Rle instance of length 997560 with 5679 runs
Lengths:  63 65 175 66 26 259 176 170 127 157 ...
Values :  0 2 0 4 6 0 4 0 6 0 ...
> x > 0
'logical' Rle instance of length 997560 with 4654 runs
Lengths:  63 65 175 92 259 176 170 127 157 144 ...
Values : FALSE TRUE FALSE TRUE FALSE TRUE ...
> xVector <- as.vector(x)
> print(object.size(x), units = "Kb")
44.8 Kb
> print(object.size(xVector), units = "Kb")
3896.7 Kb
> mean(x)
[1] 1.156386
> mean(xVector)
[1] 1.156386
```

IRanges

- The Rle class supports many methods to allow manipulation of coverage information

```
> head(runLength(x))
[1] 63 65 175 66 26 259
> head(runValue(x))
[1] 0 2 0 4 6 0
> nrun(x)
[1] 5679

> table(x)
  0      1      2      3      4      5      6      7      8
648340 76984 70263 59029 49438 37598 28316 17407 10185
> quantile(x)
 0%  25%  50%  75% 100%
  0    0    0    2    8

> subseq(x, 1, 500000)
'integer' Rle instance of length 500000 with 2849 runs
Lengths: 63 65 175 66 26 259 176 170 127 157 ...
Values : 0 2 0 4 6 0 4 0 6 0 ...

> system.time(x[1:500000])
 user system elapsed
0.444  0.056  0.502

> system.time(subseq(x, 1, 500000))
 user system elapsed
0.000  0.000  0.001
```

IRanges

- XDataFrame instances provide a container for sequences of equal length

```
> xdf <- XDataFrame(forward = x, reverse = rev(x))
> xdf
XDataFrame object with 997560 rows and 2 columns.

> xdf[["forward"]]
'integer' Rle instance of length 997560 with 5679 runs
Lengths:  63 65 175 66 26 259 176 170 127 157 ...
Values :  0 2 0 4 6 0 4 0 6 0 ...

> xdf[["reverse"]]
'integer' Rle instance of length 997560 with 5679 runs
Lengths:  134 75 172 209 72 89 229 66 166 155 ...
Values :  5 0 4 0 6 4 0 7 0 3 ...

> xdf[["forward"]] + xdf[["reverse"]]
'integer' Rle instance of length 997560 with 11327 runs
Lengths:  63 65 6 75 94 66 12 14 195 64 ...
Values :  5 7 5 0 4 8 10 6 0 6 ...

> cor(xdf[["forward"]], xdf[["reverse"]])
[1] -0.01158099
```

IRanges

- Views instances provide selection of subsequences

```
> Views(x, r)
Views on a 997560-length Rle subject
```

```
views:
```

```
   start end width
[1]   400 419    20 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[2]   350 367    18 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
[3]   300 319    20 [0 0 0 0 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
[4]   250 267    18 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
> head(Views(x, xdf[["reverse"]] > 0))
Views on a 997560-length Rle subject
```

```
views:
```

```
   start  end width
[1]     1  134   134 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[2]   210  381   172 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[3]   591  751   161 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[4]   981 1046    66 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 6 6 6 6 6 6]
[5]  1213 1367   155 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[6]  1401 1585   185 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```


IRanges

Per-range computations

```
> x
'integer' Rle instance of length 997560 with 5679 runs
Lengths:  63 65 175 66 26 259 176 170 127 157 ...
Values :  0 2 0 4 6 0 4 0 6 0 ...

> r
IRanges object:
  start end width
1   400 419    20
2   350 367    18
3   300 319    20
4   250 267    18

> aggregate(x, r, sum)
[1]  0 72 64  0
> s1 <- slice(x, lower = 1)
> length(s1)
[1] 2327
> head(viewSums(s1))
[1] 130 420 704 762 144 332
> head(viewMaxs(s1))
[1] 2 6 4 6 1 4
> head(viewApply(s1, median))
[1] 2 4 4 6 1 4
```

Biostrings

Basic containers

- Single sequence: XString (virtual class) and its direct extensions BString, DNASTring, RNASTring and AAString

```
> b <- BString("I am a BString object")
> b
 21-letter "BString" instance
seq: I am a BString object
> d <- DNASTring("TTGAAaa-ctc-N") # case will be ignored
> d
 13-letter "DNASTring" instance
seq: TTGAAAA-CTC-N
> alphabet(d)
[1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+"
> 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"
```

```
> DNASTring("this is not gonna work")
Error in .charToRawPtr(x, start = start, end = end, width = width, lkup = lkup, :
  key 105 not in lookup table
```

Biostrings

Basic containers

- Set of sequences: XStringSet (virtual class) and its direct extensions BStringSet, DNAStringSet, RNAStringSet and AStringSet

```
> s <- BStringSet(row.names(mtcars))
> s
A BStringSet instance of length 32
  width seq
[1]     9 Mazda RX4
[2]    13 Mazda RX4 Wag
[3]    10 Datsun 710
[4]    14 Hornet 4 Drive
[5]    17 Hornet Sportabout
[6]     7 Valiant
[7]    10 Duster 360
[8]     9 Merc 240D
[9]     8 Merc 230
...
[24]    10 Camaro Z28
[25]    16 Pontiac Firebird
[26]     9 Fiat X1-9
[27]    13 Porsche 914-2
[28]    12 Lotus Europa
[29]    14 Ford Pantera L
[30]    12 Ferrari Dino
[31]    13 Maserati Bora
[32]    10 Volvo 142E
```

```
> width(s)
 [1]  9 13 10 14 17  7 10  9  8  8
 9
[12] 10 10 11 18 19 17  8 11 14 13
16
[23] 11 10 16  9 13 12 14 12 13 10
> s[4:1]
A BStringSet instance of length 4
  width seq
[1]    14 Hornet 4 Drive
[2]    10 Datsun 710
[3]    13 Mazda RX4 Wag
[4]     9 Mazda RX4
> s[[2]]
13-letter "BString" instance
seq: Mazda RX4 Wag
```

Biostrings

Basic containers

- Set of views on a sequence: XStringViews

```
> d <- DNASTring("TTGAAaa-ctc-N")
> v15 <- Views(d, start=-2:12, end=0:14)
> v15
Views on a 13-letter DNASTring subject
subject: TTGAAAA-CTC-N
views:
      start end width
[1]     -2   0     3 [  ]
[2]     -1   1     3 [ T ]
[3]      0   2     3 [ TT ]
[4]      1   3     3 [TTG]
[5]      2   4     3 [TGA]
[6]      3   5     3 [GAA]
[7]      4   6     3 [AAA]
[8]      5   7     3 [AAA]
[9]      6   8     3 [AA-]
[10]     7   9     3 [A-C]
[11]     8  10     3 [-CT]
[12]     9  11     3 [CTC]
[13]    10  12     3 [TC-]
[14]    11  13     3 [C-N]
[15]    12  14     3 [-N ]
```

```
> start(v15)
[1] -2 -1  0  1  2  3  4  5  6  7  8
[12]  9 10 11 12
> end(v15)
[1]  0  1  2  3  4  5  6  7  8  9 10
[12] 11 12 13 14
> width(v15)
[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
> v15[start(v15) < 1]
Views on a 13-letter DNASTring subject
subject: TTGAAAA-CTC-N
views:
      start end width
[1]     -2   0     3 [  ]
[2]     -1   1     3 [ T ]
[3]      0   2     3 [ TT ]
> v15[[4]]
3-letter "DNASTring" instance
seq: TTG
> v15[[3]]
Error in v15[[3]] : view is out of limits
```

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...TTTAAACTCGTATCGGTAAACCAACCCTTGGAT
```

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

```
> orfs <- read.DNAStringSet(file, "fasta")
Read 450 items
> orfs
  A DNAStringSet instance of length 7
   width seq
[1]  5573 ACTTGTAATATATCTTTTA...ATCGACCTTATTGTTGATAT YAL001C TFC3 SGDI...
[2]  5825 TTCCAAGGCCGATGAATTCG...AAATTTTTTTCTATTCTCTT YAL002W VPS8 SGDI...
[3]  2987 CTTTCATGTCAGCCTGCACTT...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...AAGGAAGAAAAAAAAAATCAC YAL009W SP07 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
[1] 5573 ACUUGUAAAUAUAUCUUUUA...AUCGACCUUAUUGUUGAUUAU YAL001C TFC3 SGDI...
[2] 5825 UUCCAAGGCCGAUGAAUUCG...AAAUUUUUUUUCUAUUCUCUU 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 CAAGAUAAUGUCAAGUUAG...AAGGAAGAAAAAAAAAUCAC YAL009W SP07 SGDI...
> RNAStringSet(orfs, end=12)
A RNAStringSet instance of length 7
  width seq
[1] 12 ACUUGUAAAUAU
[2] 12 UUCCAAGGCCGA
[3] 12 CUUCAUGUCAGC
[4] 12 CACUCAUAUCGG
[5] 12 AGAGAAAGAGUU
[6] 12 GUGUCCGGGCCU
[7] 12 CAAGAUAAUGUC
  names
YAL001C TFC3 SGDI...
YAL002W VPS8 SGDI...
YAL003W EFB1 SGDI...
YAL005C SSA1 SGDI...
YAL007C ERP2 SGDI...
YAL008W FUN14 SGD...
YAL009W SP07 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 occurrences (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 literally 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:

TTTACGAGATTTATGATGATCGGATTATAACAAG

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

pattern:

ATGAT

subject:

ACGGAATTAGACCAT
TTTACGAGATTTATGATGATCGGATTATAACAAG
...
TGGACAGGTACGTAGGATGCGGTTA

Biostrings

String matching

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

patterns:

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

subject:

```
TTTACGAGATTTATGATGATCGGATTATAACAAG
```

- The set of patterns (aka the *dictionary*) needs to be preprocessed 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
TTTACGAGATTTATGATGATCGGATTATAACAAG
...
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$sequence)
> dict0
A DNASTringSet instance of length 201800
      width seq
 [1]    25 TGGCTCCTGCTGAGGTCCCCTTTCC
 [2]    25 GGCTGTGAATTCCTGTACATATTTCC
 [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 AACTGGAACCTTGTGCTCCTTGTC
[201794] 25 CAATTCGATGATGAGTCAGCAACTG
[201795] 25 GACTTTCTGAGGAGATGGATAGCCT
[201796] 25 AGATGGATAGCCTTCTGTCAAAGCA
[201797] 25 ATAGCCTTCTGTCAAAGCATCATCT
[201798] 25 TTCTGTCAAAGCATCATCTCAACAA
[201799] 25 CAAAGCATCATCTCAACAAGCCCTC
[201800] 25 GTGCTCCTTGTC AACAGCGCACCCA
```


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	TTAACAACTA
...		
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(pdickt, 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:

- **pairwiseAlignment()**: solves (Needleman-Wunsch) global alignment, (Smith-Waterman) local alignment, and (ends-free) overlap alignment problems
- **matchPWM()**: find motifs described by a Position Weight Matrix (PWM)
- **matchLRPatterns()**: matches specified by a left pattern, a right pattern and a maximum distance them
- **trimLRPatterns()**: trims left and/or right flanking patterns from one or more subjects
- **matchProbePair()**: find amplicons given by a pair of primers (simulate PCR)
- **findPalindromes()** / **findComplementedPalindromes()**

Support for indels:

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

BSgenome data packages

- Full genome sequences stored in Biostrings containers / 1 genome per package
- 13 organisms / 19 packages in the current release (BioC 2.3)
- 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.Ecoli.NCBI.20080805"
[12] "BSgenome.Ggallus.UCSC.galGal3"
[13] "BSgenome.Hsapiens.UCSC.hg17"
[14] "BSgenome.Hsapiens.UCSC.hg18"
[15] "BSgenome.Mmusculus.UCSC.mm8"
[16] "BSgenome.Mmusculus.UCSC.mm9"
[17] "BSgenome.Ptroglodytes.UCSC.panTro2"
[18] "BSgenome.Rnorvegicus.UCSC.rn4"
[19] "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$chr1
161106243-letter "MaskedDNAString" instance (# for masking)
seq: TACCCCACTCACACTTATGGATAGATCAACTAAACA...TCCATTTTAGTTTATTTTTTTTGTATGGTAGAATACT
masks:
  maskedwidth  maskedratio  active  names  desc
1      11225171 6.967558e-02  TRUE  AGAPS  assembly gaps
2         1098 6.815378e-06  TRUE   AMB  intra-contig ambiguities
3      72217189 4.482582e-01 FALSE   RM  RepeatMasker
4        314874 1.954449e-03 FALSE  TRF  Tandem Repeats Finder [period<=12]
all masks together:
  maskedwidth  maskedratio
      83443591  0.5179414
all active masks together:
  maskedwidth  maskedratio
      11226269  0.0696824
> class(Btaurus$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"
```


BSgenome data packages

Computations on masked sequences

```
> chr1
 161106243-letter "MaskedDNAString" instance (# for masking)
seq: TACCCCACTCACACTTATGGATAGATCAACTAAACA...TCCATTTTAGTTTATTTTTTTTGTATGGTAGAATACT
masks:
  maskedwidth  maskedratio  active  names  desc
1     11225171 6.967558e-02  TRUE  AGAPS  assembly gaps
2         1098 6.815378e-06  FALSE  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
 11225171     0.06967558
> alphabetFrequency(chr1)
      A      C      G      T      M      R      W      S      Y
45035268 30051163 29999849 44793694      0      0      0      0      0
      K      V      H      D      B      N      -      +
      0      0      0      0      0    1098      0      0
> countPattern("ACYTANCAGT", chr1, fixed=FALSE)
[1] 1429
> active(masks(chr1))["AMB"] <- TRUE
> countPattern("ACYTANCAGT", chr1, fixed=FALSE)
[1] 1415
> active(masks(chr1)) <- TRUE
> countPattern("ACYTANCAGT", chr1, fixed=FALSE)
[1] 632
```

BSgenome data packages

More computations on masked sequences

```
> chr1
  161106243-letter "MaskedDNAString" instance (# for masking)
seq: #####GTTTATTTTTTTGTATGGTAGAATACT
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  TRUE   RM   RepeatMasker
4       314874  1.954449e-03  TRUE  TRF  Tandem Repeats Finder [period<=12]
all masks together:
  maskedwidth  maskedratio
  83443591    0.5179414
> as(chr1, "XStringViews")
Views on a 161106243-letter DNAString subject
subject: TACCCACTCACACTTATGGATAGATCAACTAAC...CATTTTAGTTTATTTTTTTGTATGGTAGAATACT
views:
      start      end  width
 [1]      60     142     83 [AAAAAAAAAACATGTATCTAT...TAAAAGGCGTTAAATTTATC]
 [2]     195     521    327 [TATAGAGATAAAATTAATTT...AAACAAATAAAATAATATTT]
 [3]     549     552     4 [TTAA]
 [4]    1104    1251    148 [AAAATAAAAAATGCTAAGAGA...ATCTTAAAGTTATTGAAAAT]
 [5]    1711    1719     9 [GCTGCTATG]
 ...
 [231223] 161104479 161104598    120 [ACCAGTTTAATAATGTGCACT...TCTAAAAACAAAACAGGGGC]
 [231224] 161104764 161105064    301 [AAAACAAATATTAATAATAA...TTAGCTGAGGAAAATCCA]
 [231225] 161105160 161105177     18 [GCCCCACCCCCACCCCC]
 [231226] 161105497 161105785    289 [ATATCCTCTTTTGAATAGACT...AAAATGAACTGTCTTCGAT]
 [231227] 161106217 161106243     27 [GTTTATTTTTTTGTATGGTAGAATACT]
```