Efficient genome searching with Biostrings and the BSgenome data packages

Hervé Pagès

August 2, 2008

Contents

1 The Biostrings-based genome data packages .......................... 1
2 Finding an arbitrary nucleotide pattern in a chromosome .......... 2
3 Finding an arbitrary nucleotide pattern in an entire genome ...... 6
4 Some precautions when using matchPattern ........................... 11
5 Masking the chromosome sequences .................................... 12
6 Hard masking ..................................................................... 17
7 Injecting known SNPs in the chromosome sequences .............. 17
8 Finding all the patterns of a constant width dictionary in an entire genome .................................................. 17
9 Session info ....................................................................... 19

1 The Biostrings-based genome data packages

The Bioconductor project provides data packages that contain the full genome sequences of a given organism. These packages are called Biostrings-based genome data packages because the sequences they contain are stored in some of the basic containers defined in the Biostrings package, like the DNAString, the DNAStringSet or the MaskedDNAString containers. Regardless of the particular sequence data that they contain, all the Biostrings-based genome data packages are very similar and can be manipulated in a consistent and easy way. They all require the BSgenome package in order to work properly. This package, unlike the Biostrings-based genome data packages, is a software package that provides the infrastructure needed to support them (this is why the Biostrings-based genome data packages are also called BSgenome data packages). The BSgenome package itself requires the Biostrings package.

See the man page for the available.genomes function (?available.genomes) for more information about how to get the list of all the BSgenome data packages currently available
in your version of Bioconductor (you need an internet connection so that available.genomes can query the Bioconductor package repositories).

More genomes can be added if necessary. Note that the process of making a BSgenome data package is not yet documented but you are welcome to ask for help on the bioc-devel mailing list (http://bioconductor.org/docs/mailList.html) if you need a genome that is not yet available.

2 Finding an arbitrary nucleotide pattern in a chromosome

In this section we show how to find (or just count) the occurrences of some arbitrary nucleotide pattern in a chromosome. The basic tool for this is the matchPattern (or countPattern) function from the Biostrings package.

First we need to install and load the BSgenome data package for the organism that we want to look at. In our case, we want to search chromosome I of Caenorhabditis elegans.

UCSC provides several versions of the C. elegans genome: ce1, ce2 and ce4. These versions correspond to different releases from WormBase, which are the WS100, WS120 and WS170 releases, respectively. See http://genome.ucsc.edu/FAQ/FAQreleases#release1 for the list of all UCSC genome releases and for the correspondance between UCSC versions and release names.

The BSgenome data package for the ce2 genome is BSgenome.Celegans.UCSC.ce2. Note that ce1 and ce4 are not available in Bioconductor but they could be added if there is demand for them.

See ?available.genomes for how to install BSgenome.Celegans.UCSC.ce2. Then load the package and display the single object defined in it:

```r
> library(BSgenome.Celegans.UCSC.ce2)
> ls("package:BSgenome.Celegans.UCSC.ce2")
[1] "Celegans"

> Celegans
C. elegans 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 is a BSgenome object:

> class(Celegans)

[1] "BSgenome"

attr(,"package")
[1] "BSgenome"

When displayed, some basic information about the origin of the genome is shown (organism, provider, provider version, etc...) followed by the index of single sequences and eventually an additional index of multiple sequences. Methods (adequately called accessor methods) are defined for individual access to this information:

> organism(Celegans)

[1] "Caenorhabditis elegans"

> provider(Celegans)

[1] "UCSC"

> providerVersion(Celegans)

[1] "ce2"

> seqnames(Celegans)

[1] "chrI"  "chrII"  "chrIII"  "chrIV"  "chrV"  "chrX"  "chrM"

> mseqnames(Celegans)

[1] "upstream1000"  "upstream2000"  "upstream5000"

See the man page for the BSgenome class (?BSgenome) for a complete list of accessor methods and their descriptions.

Now we are ready to display chromosome I:

> Celegans$chrI

15080483-letter "DNAString" instance
seq: GCCTAAGCTAAGCTAAGCTAAGCTAAGCTAA...TTAGGCTTAGGCTTAGGCTTAGGCTTAGGC

Note that this chrI sequence corresponds to the forward strand (aka direct or sense or positive or plus strand) of chromosome I. UCSC, and genome providers in general, don’t provide files containing the nucleotide sequence of the reverse strand (aka indirect or antisense or negative or minus or opposite strand) of the chromosomes because these sequences can be deduced from the forward sequences by taking their reverse complements. The BSgenome data packages are no exceptions: they only provide the forward strand sequence of every chromosome. See ?reverseComplement for more details about the reverse complement of a
**DNAString** object. It is important to remember that, in practice, the reverse strand sequence is almost never needed. The reason is that, in fact, a reverse strand analysis can (and should) always be transposed into a forward strand analysis. Therefore trying to compute the reverse strand sequence of an entire chromosome by applying `reverseComplement` to its forward strand sequence is almost always a bad idea. See the Finding an arbitrary nucleotide pattern in an entire genome section of this document for how to find arbitrary patterns in the reverse strand of a chromosome.

The number of bases in this sequence can be retrieved with:

```r
> chrI <- Celegans$chrI
> length(chrI)

[1] 15080483
```

Some basic stats:

```r
> afI <- alphabetFrequency(chrI)
> afI

A  C  G  T  M  R  W  S  Y  K
4838561 2697177 2693544 4851201 0  0  0  0  0  0
V  H  D  B  N  -  +
0  0  0  0  0  0  0

> sum(afI) == length(chrI)

[1] TRUE
```

Count all exact matches of pattern "ACCCAGGGC":

```r
> p1 <- "ACCCAGGGC"
> countPattern(p1, chrI)

[1] 0
```

Like most pattern matching functions in Biostrings, the `countPattern` and `matchPattern` functions support inexact matching. One form of inexact matching is to allow a few mismatching letters per match. Here we allow at most one:

```r
> countPattern(p1, chrI, max.mismatch=1)

[1] 235
```

With the `matchPattern` function, the locations of the matches are stored in an `XStringViews` object:

```r
> m1 <- matchPattern(p1, chrI, max.mismatch=1)
> m1[4:6]
```
Views on a 15080483-letter DNAString subject

subject: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...AGGCTTAGGCTTAGGCTTAGGTTAGGCTTAGGC

views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>187350</td>
<td>9</td>
<td>CCCAAGGC</td>
</tr>
<tr>
<td>2</td>
<td>213236</td>
<td>9</td>
<td>CCCAAGGGG</td>
</tr>
<tr>
<td>3</td>
<td>424133</td>
<td>9</td>
<td>CCCAAGGAC</td>
</tr>
</tbody>
</table>

> class(m1)

[1] "XStringViews"
attr("package")
[1] "Biostrings"

The `mismatch` function (new in Biostrings 2) returns the positions of the mismatching letters for each match:

> mismatch(p1, m1[4:6])

[[1]]
[1] 6

[[2]]
[1] 9

[[3]]
[1] 8

Note: The `mismatch` method is in fact a particular case of a (vectorized) alignment function where only “replacements” are allowed. Current implementation is slow but this will be addressed.

It may happen that a match is out of limits like in this example:

> p2 <- DNAString("AAGCCTAAGCCTAAGCCTAA")
> m2 <- matchPattern(p2, chrI, max.mismatch=2)
> m2[1:4]

Views on a 15080483-letter DNAString subject

subject: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...AGGCTTAGGCTTAGGCTTAGGTTAGGCTTAGGC

views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>18</td>
<td>20</td>
<td>GCCTAAGCCTAAGCCTAA</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>20</td>
<td>AAGCCTAAGCCTAAGCCTAA</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>20</td>
<td>AAGCCTAAGCCTAAGCCTAA</td>
</tr>
<tr>
<td>17</td>
<td>36</td>
<td>20</td>
<td>AAGCCTAAGCCTAAGCCTAA</td>
</tr>
</tbody>
</table>

> p2 == m2[1:4]
The list of exact matches and the list of inexact matches can both be obtained with:

```r
> m2[p2 == m2]
> m2[p2 != m2]
```

Note that the length of `m2[p2 == m2]` should be equal to `countPattern(p2, chrI, max.mismatch=0)`.

### 3 Finding an arbitrary nucleotide pattern in an entire genome

Now we want to extend our analysis to the forward and reverse strands of all the C. elegans chromosomes. More precisely, here is the analysis we want to perform:

- **The input dictionary:** Our input is a dictionary of 50 patterns. Each pattern is a short nucleotide sequence of 15 to 25 bases (As, Cs, Gs and Ts only, no Ns). It is stored in a FASTA file called "ce2dict0.fa". See the Finding all the patterns of a constant width dictionary in an entire genome section of this document for a very efficient way to deal with the special case where all the patterns in the input dictionary have the same length.

- **The target:** Our target (or subject) is the forward and reverse strands of the seven C. elegans chromosomes (14 sequences in total). We want to find and report all occurrences (or hits) of every pattern in the target. Note that a given pattern can have 0, 1 or several hits in 0, 1 or 2 strands of 0, 1 or several chromosomes.

- **Exact or inexact matching?** We are interested in exact matches only (for now).

- **The output:** We want to put the results of this analysis in a file so we can send it to our collaborators for some post analysis work. Our collaborators are not necessarily familiar with R or Bioconductor so dumping a high-level R object (like a list or a data frame) into an .rda file is not an option. For maximum portability (one of our collaborators wants to use Microsoft Excel for the post analysis) we choose to put our results in a tabulated file where one line describes one hit. The columns (or fields) of this file will be (in this order):
– seqname: the name of the chromosome where the hit occurs.
– start: an integer giving the starting position of the hit.
– end: an integer giving the ending position of the hit.
– strand: a plus (+) for a hit in the positive strand or a minus (−) for a hit in the negative strand.
– patternID: we use the unique ID provided for every pattern in the "ce2dict0.fa" file.

Let’s start by loading the input dictionary with:

```r
> ce2dict0_file <- system.file("extdata", "ce2dict0.fa", package="BSgenome")
> ce2dict0 <- read.DNAStringSet(ce2dict0_file, "fasta")
> ce2dict0
```

A DNAStringSet instance of length 50

<table>
<thead>
<tr>
<th>width</th>
<th>seq names</th>
<th>names</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>18 GCGAAACTAGGAGGCT</td>
<td>pattern01</td>
</tr>
<tr>
<td>[2]</td>
<td>25 CTGTTAGCTAATTAAAAAATAAT</td>
<td>pattern02</td>
</tr>
<tr>
<td>[3]</td>
<td>24 ACTACCCACCCAAAATTTAGATTC</td>
<td>pattern03</td>
</tr>
<tr>
<td>[5]</td>
<td>25 TCTCTCTGCTTTTTGTTGTTCTTT</td>
<td>pattern05</td>
</tr>
<tr>
<td>[6]</td>
<td>16 AACAAATTATCTCTATAAT</td>
<td>pattern06</td>
</tr>
<tr>
<td>[7]</td>
<td>22 GGTGTGGAGAGTGATGCACGT</td>
<td>pattern07</td>
</tr>
<tr>
<td>[8]</td>
<td>21 TTTATGAAACCCAGCAACTC</td>
<td>pattern08</td>
</tr>
<tr>
<td>[9]</td>
<td>18 TACTGAAACTCCCCGCAGGAG</td>
<td>pattern09</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[42]</td>
<td>22 AAAATAAGATTCTATAAAAATA</td>
<td>pattern42</td>
</tr>
<tr>
<td>[43]</td>
<td>15 GTTTTAAATTTCTAAATT</td>
<td>pattern43</td>
</tr>
<tr>
<td>[44]</td>
<td>18 CATCGTTTCAACCGTTGCG</td>
<td>pattern44</td>
</tr>
<tr>
<td>[45]</td>
<td>19 TAAAAATCAAACCTTTTTTG</td>
<td>pattern45</td>
</tr>
<tr>
<td>[46]</td>
<td>24 TTTGAAACAAAGCATGTCTAACTA</td>
<td>pattern46</td>
</tr>
<tr>
<td>[47]</td>
<td>20 TAAAGCAATTTGGAATTATATAT</td>
<td>pattern47</td>
</tr>
<tr>
<td>[48]</td>
<td>19 AAGGACCAGGATTGCCACG</td>
<td>pattern48</td>
</tr>
<tr>
<td>[49]</td>
<td>24 AAAATAACTGCCTAAAAACACACTA</td>
<td>pattern49</td>
</tr>
<tr>
<td>[50]</td>
<td>22 AAAATGGCCCGGACATTTAAAAAG</td>
<td>pattern50</td>
</tr>
</tbody>
</table>

Here is how we can write the functions that will perform our analysis:

```r
> writeHits <- function(seqname, matches, strand, file=",", append=FALSE) {
+   if (file.exists(file) && !append)
+     warning("existing file ", file, " will be overwritten with 'append=FALSE'")
+   if (!file.exists(file) && append)
+     warning("new file ", file, " will have no header with 'append=TRUE'")
+   hits <- data.frame(seqname=rep.int(seqname, length(matches)),
```

```r
```
Some important notes about the implementation of the runAnalysis1 function:

- `subject <- Celegans[[seqname]]` is the code that actually loads a chromosome sequence into memory. After it’s not needed anymore, the sequence is removed from memory by calling `unload(Celegans, seqname)`. Using only one sequence at a time and unloading it after it’s not needed anymore is a good practice to avoid memory allocation problems on a machine with a limited amount of memory. For example, loading all the human chromosome sequences in memory would require more than 3GB of memory!

- We have 2 nested `for` loops: the outer loop walks thru the target (7 chromosomes) and the inner loop walks thru the set of patterns. Doing the other way around would be very
inefficient, especially with a bigger number of patterns because this would require to load each chromosome sequence into memory as many times as the number of patterns. Trying to minimize the number of loading/unloading cycles for each target sequence during an analysis is good practice because this cycle takes a long time. The runAnalysis1 does this only once for each target sequence.

- We find the matches in the minus strand (minus_matches) by first taking the reverse complement of the current pattern (with rcpattern <- reverseComplement(pattern)) and NOT by taking the reverse complement of the current subject.

Now we are ready to run the analysis and put the results in the "ce2dict0_ana1.txt" file:

```r
> runAnalysis1(ce2dict0, outfile="ce2dict0_ana1.txt")
```

```
Target: ce2 chromosomes chrI, chrII, chrIII, chrIV, chrV, chrX, chrM
>>> Finding all hits in chromosome chrI ...
   >>> DONE
>>> Finding all hits in chromosome chrII ...
   >>> DONE
>>> Finding all hits in chromosome chrIII ...
   >>> DONE
>>> Finding all hits in chromosome chrIV ...
   >>> DONE
>>> Finding all hits in chromosome chrV ...
   >>> DONE
>>> Finding all hits in chromosome chrX ...
   >>> DONE
>>> Finding all hits in chromosome chrM ...
   >>> DONE
```

Here is some very simple example of post analysis:

- Get the total number of hits:

```r
> hits1 <- read.table("ce2dict0_ana1.txt", header=TRUE)
> nrow(hits1)

[1] 79
```

- Get the number of hits per chromosome:

```r
> table(hits1$seqname)

chrI chrII chrIII chrIV chrM chrV chrX
 11   5   16   8   8  15  16
```

- Get the number of hits per pattern:
> hits1_table <- table(hits1$patternID)
> hits1_table

<table>
<thead>
<tr>
<th>pattern01</th>
<th>pattern02</th>
<th>pattern03</th>
<th>pattern04</th>
<th>pattern06</th>
<th>pattern07</th>
<th>pattern08</th>
<th>pattern09</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern10</th>
<th>pattern11</th>
<th>pattern12</th>
<th>pattern13</th>
<th>pattern14</th>
<th>pattern15</th>
<th>pattern16</th>
<th>pattern17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern18</th>
<th>pattern19</th>
<th>pattern20</th>
<th>pattern21</th>
<th>pattern22</th>
<th>pattern23</th>
<th>pattern24</th>
<th>pattern25</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern26</th>
<th>pattern27</th>
<th>pattern28</th>
<th>pattern29</th>
<th>pattern30</th>
<th>pattern31</th>
<th>pattern32</th>
<th>pattern33</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern34</th>
<th>pattern35</th>
<th>pattern36</th>
<th>pattern37</th>
<th>pattern38</th>
<th>pattern39</th>
<th>pattern40</th>
<th>pattern41</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern42</th>
<th>pattern43</th>
<th>pattern44</th>
<th>pattern45</th>
<th>pattern46</th>
<th>pattern47</th>
<th>pattern48</th>
<th>pattern49</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern50</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

- Get the pattern(s) with the higher number of hits:

> hits1_table[hit1_table == max(hits1_table)] # pattern(s) with more hits

```
        pattern21
          10
```

- Get the pattern(s) with no hits:

> setdiff(names(ce2dict0), hits1$patternID) # pattern(s) with no hits

```
[1] "pattern05"
```

- And finally a function that can be used to plot the hits:

```R
> plotGenomeHits <- function(bsgenome, seqnames, hits)
+ {
+   ## Retrieve the sequence lengths
+   seqlengths <- integer(length(seqnames))
+   for (i in seq_len(length(seqnames))) {
+     seqname <- seqnames[i]
+     seqlengths[i] <- length(bsgenome[[seqname]])
+     unload(bsgenome, seqname)
+   }
+   XMAX <- max(seqlengths)
+   YMAX <- length(seqnames)
+   plot.new()
+   plot.window(c(1, XMAX), c(0, YMAX))
+   axis(1)
+  }
```
axis(2, at=seq_len(length(seqnames)), labels=rev(seqnames), tick=FALSE, las=1)
## Plot the chromosomes
for (i in seq_len(length(seqnames)))
  lines(c(1, seqlengths[i]), c(YMAX + 1 - i, YMAX + 1 - i), type="l")
## Plot the hits
for (i in seq_len(nrow(hits))) {
  seqname <- hits$seqname[i]
  y0 <- YMAX + 1 - match(seqname, seqnames)
  if (hits$strand[i] == "+") {
    y <- y0 + 0.05
    col <- "red"
  } else {
    y <- y0 - 0.05
    col <- "blue"
  }
  lines(c(hits$start[i], hits$end[i]), c(y, y), type="l", col=col, lwd=3)
}

Plot the hits found by runAnalysis1 with:

> plotGenomeHits(Celegans, seqnames(Celegans), hits1)

4 Some precautions when using matchPattern

Improper use of matchPattern (or countPattern) can affect performance.

If needed, the matchPattern and countPattern methods convert their first argument (the pattern) to an object of the same class than their second argument (the subject) before they pass it to the subroutine that actually implements the fast search algorithm.

So if you need to reuse the same pattern a high number of times, it’s a good idea to convert it before to pass it to the matchPattern or countPattern method. This way the conversion is done only once:

> library(hgu95av2probe)
> tmpseq <- DNAStringSet(hgu95av2probe$sequence)
> someStats <- function(v)
+  {
+    GC <- DNAString("GC")
+    CG <- DNAString("CG")
+    sapply(seq_len(length(v)),
+      function(i) {
+        y <- v[[i]]
+        c(alphabetFrequency(y)[1:4],
+          GC=countPattern(GC, y),
+          CG=countPattern(CG, y))
+      }
+  }

11
5 Masking the chromosome sequences

Starting with Bioconductor 2.2, some BSgenome data packages provide built-in masks for the chromosome sequences. For example, each chromosome in BSgenome.Hsapiens.UCSC.hg18 has 3 masks on it: the mask of assembly gaps, the mask of repeat regions that were determined by the RepeatMasker software, and the mask of repeat regions that were determined by the Tandem Repeats Finder software (where only repeats with period less than or equal to 12 were kept).

> library(BSgenome.Hsapiens.UCSC.hg18)
> chrY <- Hsapiens$chrY
> chrY

57772954-letter "MaskedDNAString" instance (# for masking)
seq: CTAACCCTAACCCTAACCCTAACCCTAACCCTAACC...GGGTTGCGGTTGCTGCGGTTGCGGTTGCGGTTGCGGTTG
masks:

maskedwidth  maskedratio active names
1 32120000 0.55596949 FALSE assembly gaps
2 15884416 0.27494554 FALSE RepeatMasker
3 584387 0.01011523 FALSE Tandem Repeats Finder [period<=12]

all masks together:
maskedwidth  maskedratio
48042573 0.8315755

all active masks together:
maskedwidth  maskedratio
0 0

Note that the built-in masks are always inactive by default. When displaying a masked sequence (here a MaskedDNAString object), the masked width and masked ratio are reported for each individual mask, as well as for all the masks together, and for all the active masks together. The masked width is the total number of nucleotide positions that are masked and the masked ratio is the masked width divided by the length of the sequence.

To activate a mask, use the active replacement method in conjunction with the masks method. For example, to activate the first mask (assembly gaps), do:
> active(masks(chrY))[1] <- TRUE
> chrY

57772954-letter "MaskedDNAString" instance (# for masking)
seq: CTAACCCTAACCCTAACCCTAACCCTAACCCTAACC...GGGTGTGGGTGTGTGGGTGTGGTGTGGT
masks:
<table>
<thead>
<tr>
<th>maskedwidth</th>
<th>maskedratio</th>
<th>active</th>
<th>names</th>
</tr>
</thead>
<tbody>
<tr>
<td>32120000</td>
<td>0.55596949</td>
<td>TRUE</td>
<td>assembly gaps</td>
</tr>
<tr>
<td>15884416</td>
<td>0.27494554</td>
<td>FALSE</td>
<td>RepeatMasker</td>
</tr>
<tr>
<td>584387</td>
<td>0.01011523</td>
<td>FALSE</td>
<td>Tandem Repeats Finder [period&lt;=12]</td>
</tr>
</tbody>
</table>

all masks together:

<table>
<thead>
<tr>
<th>maskedwidth</th>
<th>maskedratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>48042573</td>
<td>0.8315755</td>
</tr>
</tbody>
</table>

all active masks together:

<table>
<thead>
<tr>
<th>maskedwidth</th>
<th>maskedratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>32120000</td>
<td>0.5559695</td>
</tr>
</tbody>
</table>

As you can see, the masked width for all the active masks together (i.e. the total number of nucleotide positions that are masked by at least one active mask) is now the same as for the first mask. This represents a masked ratio of about 55.6%.

Now when we use a mask aware function like alphabetFrequency, the masked regions of the input sequence are ignored:

> alphabetFrequency(unmasked(chrY))

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>M</th>
<th>R</th>
<th>W</th>
<th>S</th>
<th>Y</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>7667357</td>
<td>5099042</td>
<td>5153198</td>
<td>7733357</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>K</td>
<td>V</td>
<td>H</td>
<td>D</td>
<td>B</td>
<td>N</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32120000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> alphabetFrequency(chrY)

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>M</th>
<th>R</th>
<th>W</th>
<th>S</th>
<th>Y</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>7667357</td>
<td>5099042</td>
<td>5153198</td>
<td>7733357</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>H</td>
<td>D</td>
<td>B</td>
<td>N</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This output indicates that, for this chromosome, the assembly gaps correspond exactly to the regions in the sequence that were filled with the letter N.

When coercing a MaskedXString object to an XStringViews object, each non-masked region in the original sequence is converted into a view on the sequence:

> as(chrY, "XStringViews")
Views on a 57772954-letter DNAString subject
subject: CTAACCCTAACCCTAACCCTAACCCTAACCCTAA...GTGTGGGTGTGGTGTTGGTGTTGGTGTTGGTGGT
views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
<th>views</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34821</td>
<td>34821</td>
<td>[CTAACCCTAACCCTAACCCTAA...AGGTTCTCATTGAGGACAGATA]</td>
</tr>
<tr>
<td>84822</td>
<td>171384</td>
<td>86563</td>
<td>[GATCCACCCCATCTCGTCTCC...CGATCTGTCAGCTGATGCATGC]</td>
</tr>
<tr>
<td>201385</td>
<td>967557</td>
<td>766173</td>
<td>[GATCGGGGTATCCCAGCTGCTA...GTGATCCAACATGGGGAAACCC]</td>
</tr>
<tr>
<td>1017558</td>
<td>1054113</td>
<td>36556</td>
<td>[ATTAAAGAAAGAGAGACTG...GTGTTGTGCTACATGCATGCT]</td>
</tr>
<tr>
<td>1104114</td>
<td>1184234</td>
<td>80121</td>
<td>[GATTGAACCCAGCCACTCCAG...CTGGCAACAGTGACTGGAATTC]</td>
</tr>
<tr>
<td>1274235</td>
<td>2028238</td>
<td>754004</td>
<td>[ATCCACCTGGCGCTCTCCTA...GTCATAAAGAAGACAGATC]</td>
</tr>
<tr>
<td>2128239</td>
<td>8974955</td>
<td>6846717</td>
<td>[CCCTTCCAGGATGTCCTCTC...GGGATATATTAGCAAGATCT]</td>
</tr>
<tr>
<td>9024956</td>
<td>9301322</td>
<td>276367</td>
<td>[GATCATGAGCTAAGGATTTGA...TTACTTTCTAATTCAGTATGC]</td>
</tr>
<tr>
<td>9901323</td>
<td>10714553</td>
<td>813231</td>
<td>[GAATTCGTTTTCTCTGCTCCTA...CAAAGAATCTGCAAGATTC]</td>
</tr>
<tr>
<td>11214554</td>
<td>11253954</td>
<td>39401</td>
<td>[GACACATCCACAAAGAAGTTTCT...CCATTCTTTCCACTGGAATTC]</td>
</tr>
<tr>
<td>11653955</td>
<td>12208578</td>
<td>554624</td>
<td>[GATCACATTCTTTACTATT...TAATGGAATGGGAATGGAATTC]</td>
</tr>
<tr>
<td>12308579</td>
<td>22310816</td>
<td>10002238</td>
<td>[GAATTCATTCGAATAGAATTGA...TTCAAAAACTTTATGGAATTC]</td>
</tr>
<tr>
<td>22360817</td>
<td>27228749</td>
<td>4867933</td>
<td>[AAGCTTTGGCTATTATATCTC...GAGTGTGCGACAGTGAATTC]</td>
</tr>
<tr>
<td>57228750</td>
<td>57327044</td>
<td>98295</td>
<td>[GAATTCATTCGGATCATTCCACT...CCATGGCAATTTCAATGGAATTC]</td>
</tr>
<tr>
<td>57377045</td>
<td>57772954</td>
<td>395910</td>
<td>[GAATTCACATTATTCTTTGGT...GTTGTGTTGTGCTGGAATTC]</td>
</tr>
</tbody>
</table>

This can be used in conjunction with the `gaps` method to see the gaps between the views i.e. the masked regions themselves:

```r
> gaps(as(chrY, "XStringViews"))
```

To extract the sizes of the assembly gaps:

```r
> width(gaps(as(chrY, "XStringViews")))
```

```r
[1] 50000 30000 50000 50000 90000 100000 50000 600000
[9] 500000 400000 100000 50000 30000000 50000
```

Note that, if applied directly to `chrY`, `gaps` returns a `MaskedDNAString` object with a single mask masking the regions that are not masked in the original object:

```r
> gaps(chrY)
```

57772954-letter "MaskedDNAString" instance (# for masking)

seq: ###########################################################################

masks:

<table>
<thead>
<tr>
<th>maskedwidth</th>
<th>maskedratio</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25652954</td>
<td>0.4440305</td>
</tr>
</tbody>
</table>

```r
> alphabetFrequency(gaps(chrY))
```

```
A 0
c 0
g 0
t 0
m 0
r 0
w 0
s 0
```

```
y 1
k 25652954
v 0.4440305
h 0
```
In fact, for any \texttt{MaskedDNAString} object, the following should always be \texttt{TRUE}, whatever the masks are:

\begin{verbatim}
> af0 <- alphabetFrequency(unmasked(chrY))
> af1 <- alphabetFrequency(chrY)
> af2 <- alphabetFrequency(gaps(chrY))
> all(af0 == af1 + af2)
[1] TRUE
\end{verbatim}

With all \texttt{chrY} masks active:

\begin{verbatim}
> active(masks(chrY)) <- TRUE
> af1 <- alphabetFrequency(chrY)
> af1

A C G T M R W S Y K
2978843 1871661 1883042 2996835 0 0 0 0 0 0
V H D B N - +
0 0 0 0 0 0 32120000 0

> gaps(chrY)

57772954-letter "MaskedDNAString" instance (# for masking)
seq: CTAACCCTAACCCTAACCCTAACCCTAACCCTAACC...GGGTGTGGGTGTGTGGGTGTGGTGTGGTGTGGTGGGTGGTGG#
masks:
maskedwidth maskedratio active
1 9730381 0.1684245 TRUE

> af2 <- alphabetFrequency(gaps(chrY))
> af2

A C G T M R W S
4688514 3227381 3270156 4736522 0 0 0 0
Y K V H D B N -
0 0 0 0 0 0 32120000 0
+
0

> all(af0 == af1 + af2)
[1] TRUE
\end{verbatim}
Now let’s compare three different ways of finding all the occurrences of the "CANNGT" consensus sequence in chrY. TheNs in this pattern need to be treated as wildcards i.e. they must match any letter in the subject.

Without the mask feature, the first way to do it would be to use the fixed=FALSE option in the call to \texttt{matchPattern} (or \texttt{countPattern}):

\begin{verbatim}
> Ebox <- "CANNGT"
> active(masks(chrY)) <- FALSE
> countPattern(Ebox, chrY, fixed=FALSE)
[1] 32224073
\end{verbatim}

The problem with this method is that theNs in the subject are also treated as wildcards hence the abnormally high number of matches. A better method is to specify the side of the matching problem (i.e. \texttt{pattern} or \texttt{subject}) where theNs should be treated as wildcards:

\begin{verbatim}
> countPattern(Ebox, chrY, fixed=c(pattern=FALSE,subject=TRUE))
[1] 104132
\end{verbatim}

Finally, \texttt{countPattern} being mask aware, this can be achieved more efficiently by simply masking the assembly gaps:

\begin{verbatim}
> active(masks(chrY))[[1]] <- TRUE
> countPattern(Ebox, chrY, fixed=FALSE)
[1] 104132
\end{verbatim}

Note that with chromosomes that haveNs outside the assembly gaps like chromosome 2, you need to use \texttt{fixed=c(pattern=FALSE,subject=TRUE)} even when the mask of assembly gaps is active:

\begin{verbatim}
> chr2 <- Hsapiens$chr2
> active(masks(chr2))[[1]] <- TRUE
> alphabetFrequency(chr2)

A C G T M R W S
70967972 47802677 47841066 71098079 0 0 0 0
Y K V H D B N -
0 0 0 0 0 0 2855 0
+
0

> countPattern(Ebox, chr2, fixed=FALSE)
[1] 942474

> countPattern(Ebox, chr2, fixed=c(pattern=FALSE,subject=TRUE))
\end{verbatim}
Or, alternatively, you could mask the Ns yourself with the `maskMotif` function:

```r
> chr2_noNs <- maskMotif(unmasked(chr2), "N")
> alphabetFrequency(chr2_noNs)

A   C   G   T   M   R   W   S
70967972 47802677 47841066 71098079 0   0   0   0
Y   K   V   H   D   B   N   -
  0   0   0   0   0   0   0   0
```

```r
> countPattern(Ebox, chr2_noNs, fixed=FALSE)
```

Note that not all functions that work with an `XString` input are *mask aware* but more will be added in the near future. However, there is no plan to do this in a systematic way since the usefulness of supporting masks depends on the function itself. For example it’s clear that making `dinucleotideFrequency` *mask aware* is going to provide a lot of convenience, but doing so for `reverseComplement` would not really make sense from a biological point of view. Or maybe it will, but we haven’t seen any use case for this at the moment. Anyway, most of the times there is a systematic way to exclude some arbitrary regions from an analysis without having to use *mask aware* functions. This is described below in the *Hard masking* section.

6  **Hard masking**

coming soon...

7  **Injecting known SNPs in the chromosome sequences**

coming soon...

8  **Finding all the patterns of a constant width dictionary in an entire genome**

The `matchPDict` function can be used instead of `matchPattern` for the kind of analysis described in the *Finding an arbitrary nucleotide pattern in an entire genome* section but it will be much faster (between 100x and 10000x faster depending on the size of the input dictionary). Note that a current limitation of `matchPDict` is that it only works with a dictionary of DNA patterns where all the patterns have the same number of nucleotides (constant width dictionary). See `?matchPDict` for more information.

Here is how our `runAnalysis1` function can be modified in order to use `matchPDict` instead of `matchPattern`:
```r
> runOneStrandAnalysis <- function(dict0, bsgenome, seqnames, strand,
+     outfile="", append=FALSE)
+ {
+     cat("\nTarget: strand", strand, "of", providerVersion(bsgenome),
+         "chromosomes", paste(seqnames, collapse=" ", ", ", \\
+     )
+     if (strand == "-"")
+         dict0 <- reverseComplement(dict0)
+     pdict <- PDict(dict0)
+     for (seqname in seqnames) {
+         subject <- bsgenome[[seqname]]
+         cat(">>> Finding all hits in strand", strand, "of chromosome", seqname, ", ...
+             ")
+         mindex <- matchPDict(pdict, subject)
+         matches <- extractAllMatches(subject, mindex)
+         writeHits(seqname, matches, strand, file=outfile, append=append)
+         append <- TRUE
+         cat(">>> DONE\n")
+         unload(bsgenome, seqname)
+     }
+ }
> runAnalysis2 <- function(dict0, outfile="")
+ {
+     library(BSgenome.Celegans.UCSC.ce2)
+     seqnames <- seqnames(Celegans)
+     runOneStrandAnalysis(dict0, Celegans, seqnames, "+", outfile=outfile, append=FALSE)
+     runOneStrandAnalysis(dict0, Celegans, seqnames, "-", outfile=outfile, append=TRUE)
+ }

Remember that matchPDict only works if all the patterns in the input dictionary have
the same length so for this 2nd analysis, we will truncate the patterns in ce2dict0 to 15
nucleotides:

> ce2dict0cw15 <- DNAStringSet(ce2dict0, end=15)

Now we can run this 2nd analysis and put the results in the "ce2dict0cw15_ana2.txt"
file:

> runAnalysis2(ce2dict0cw15, outfile="ce2dict0cw15_ana2.txt")

Target: strand + of ce2 chromosomes chrI, chrII, chrIII, chrIV, chrV, chrX, chrM
  >>> Finding all hits in strand + of chromosome chrI ...
  >>> DONE
  >>> Finding all hits in strand + of chromosome chrII ...
  >>> DONE
  >>> Finding all hits in strand + of chromosome chrIII ...
  >>> DONE
  >>> Finding all hits in strand + of chromosome chrIV ...
  >>> DONE
```

Finding all hits in strand + of chromosome chrV ...
>>> DONE
Finding all hits in strand + of chromosome chrX ...
>>> DONE
Finding all hits in strand + of chromosome chrM ...
>>> DONE

Target: strand - of ce2 chromosomes chrI, chrII, chrIII, chrIV, chrV, chrX, chrM
Finding all hits in strand - of chromosome chrI ...
>>> DONE
Finding all hits in strand - of chromosome chrII ...
>>> DONE
Finding all hits in strand - of chromosome chrIII ...
>>> DONE
Finding all hits in strand - of chromosome chrIV ...
>>> DONE
Finding all hits in strand - of chromosome chrV ...
>>> DONE
Finding all hits in strand - of chromosome chrX ...
>>> DONE
Finding all hits in strand - of chromosome chrM ...
>>> DONE

9 Session info

> sessionInfo()

R version 2.7.1 (2008-06-23)
x86_64-unknown-linux-gnu

locale:
LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_MESSAGES=en_US;LC_PAPER=en_US;LC_NAME=C;LC_ADDRESS=C;LC_TELEPHONE=C;LC_MEASUREMENT=en_US;LC_IDENTIFICATION=C

attached base packages:
[1] tools stats graphics grDevices utils datasets methods
[8] base

other attached packages:
[1] BSgenome.Hsapiens.UCSC.hg18_1.3.7 hgu95av2probe_2.2.0
[3] matchprobes_1.12.0              affy_1.18.2
[5] preprocessCore_1.2.1           affyio_1.8.1
[7] BSgenome.Celegans.UCSC.ce2_1.3.4 BSgenome_1.8.4
[9] Biostrings_2.8.17              Biobase_2.0.1