MultipleAlignment Objects

Marc Carlson
Bioconductor Core Team
Fred Hutchinson Cancer Research Center
Seattle, WA

May 2, 2019

Contents

1 Introduction

The *DNAMultipleAlignment* *RNAMultipleAlignment* and *AAMultipleAlignment* classes allow users to represent groups of aligned DNA, RNA or amino acid sequences as a single object. The frame of reference for aligned sequences is static, so manipulation of these objects is confined to be non-destructive. In practice, this means that these objects contain slots to mask ranges of rows and columns on the original sequence. These masks are then respected by methods that manipulate and display the objects, allowing the user to remove or expose columns and rows without invalidating the original alignment.

2 Creation and masking

To create a *MultipleAlignment*, call the appropriate read function to read in and parse the original alignment. There are functions to read clustaW, Phylip and Stolkholm data formats.

```
> library(Biostrings)
> origMAlign <-
+   readDNAMultipleAlignment(filepath =
+                            system.file("extdata",
+                                        "msx2_mRNA.aln",
+                                        package="Biostrings"),
+                            format="clustal")
> phylipMAlign <-
+   readAAMultipleAlignment(filepath =
+                           system.file("extdata",
```

```
+                                        "Phylip.txt",
+                                        package="Biostrings"),
+                            format="phylip")
```

Rows can be renamed with `rownames`.

```
> rownames(origMAlign)
```

```
[1] "gi|84452153|ref|NM_002449.4|"   "gi|208431713|ref|NM_001135625."
[3] "gi|118601823|ref|NM_001079614." "gi|114326503|ref|NM_013601.2|"
[5] "gi|119220589|ref|NM_012982.3|"  "gi|148540149|ref|NM_001003098."
[7] "gi|45383056|ref|NM_204559.1|"   "gi|213515133|ref|NM_001141603."
```

```
> rownames(origMAlign) <- c("Human","Chimp","Cow","Mouse","Rat",
+                           "Dog","Chicken","Salmon")
> origMAlign
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                  names
[1] -----TCCCGTCTCCGCAGCAAAAAA...TCACAATTAAAAAAAAAAAAAAAA Human
[2] -------------------------...------------------------- Chimp
[3] -------------------------...------------------------- Cow
[4] ----------------------AAAA...------------------------- Mouse
[5] -------------------------...------------------------- Rat
[6] -------------------------...------------------------- Dog
[7] --------------CGGCTCCGCAGC...------------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------- Salmon
```

To see a more detailed version of your *MultipleAlignment* object, you can use the *detail* method, which will show the details of the alignment interleaved and without the rows and columns that you have masked out.

```
> detail(origMAlign)
```

Applying masks is a simple matter of specifying which ranges to hide.

```
> maskTest <- origMAlign
> rowmask(maskTest) <- IRanges(start=1,end=3)
> rowmask(maskTest)
```

```
NormalIRanges object with 1 range and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         1         3         3
```

```
> maskTest
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                  names
[1] ###########################...######################### Human
[2] ###########################...######################### Chimp
[3] ###########################...######################### Cow
[4] ----------------------AAAA...------------------------- Mouse
[5] -------------------------...------------------------- Rat
[6] -------------------------...------------------------- Dog
[7] --------------CGGCTCCGCAGC...------------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------- Salmon
```

```
> colmask(maskTest) <- IRanges(start=c(1,1000),end=c(500,2343))
> colmask(maskTest)

NormalIRanges object with 2 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         1       500       500
  [2]      1000      2343      1344

> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                    names
[1] ##########################...######################### Human
[2] ##########################...######################### Chimp
[3] ##########################...######################### Cow
[4] ##########################...######################### Mouse
[5] ##########################...######################### Rat
[6] ##########################...######################### Dog
[7] ##########################...######################### Chicken
[8] ##########################...######################### Salmon
```

Remove row and column masks by assigning `NULL`:

```
> rowmask(maskTest) <- NULL
> rowmask(maskTest)

NormalIRanges object with 0 ranges and 0 metadata columns:
       start       end     width
   <integer> <integer> <integer>

> colmask(maskTest) <- NULL
> colmask(maskTest)

NormalIRanges object with 0 ranges and 0 metadata columns:
       start       end     width
   <integer> <integer> <integer>

> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                    names
[1] -----TCCCGTCTCCGCAGCAAAAAA...TCACAATTAAAAAAAAAAAAAAAAA Human
[2] -------------------------...------------------------- Chimp
[3] -------------------------...------------------------- Cow
[4] ----------------------AAAA...------------------------- Mouse
[5] -------------------------...------------------------- Rat
[6] -------------------------...------------------------- Dog
[7] --------------CGGCTCCGCAGC...------------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------- Salmon
```

When setting a mask, you might want to specify the rows or columns to keep, rather than to hide. To do that, use the *invert* argument. Taking the above example, we can set the exact same masks as before by specifying their inverse and using `invert=TRUE`.

```
> rowmask(maskTest, invert=TRUE) <- IRanges(start=4,end=8)
> rowmask(maskTest)

NormalIRanges object with 1 range and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         1         3         3

> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                  names
[1] ###########################...######################## Human
[2] ###########################...######################## Chimp
[3] ###########################...######################## Cow
[4] ----------------------AAAA...------------------------ Mouse
[5] --------------------------...------------------------ Rat
[6] --------------------------...------------------------ Dog
[7] --------------CGGCTCCGCAGC...------------------------ Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------ Salmon

> colmask(maskTest, invert=TRUE) <- IRanges(start=501,end=999)
> colmask(maskTest)

NormalIRanges object with 2 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         1       500       500
  [2]      1000      2343      1344

> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                  names
[1] ###########################...######################## Human
[2] ###########################...######################## Chimp
[3] ###########################...######################## Cow
[4] ###########################...######################## Mouse
[5] ###########################...######################## Rat
[6] ###########################...######################## Dog
[7] ###########################...######################## Chicken
[8] ###########################...######################## Salmon
```

In addition to being able to invert these masks, you can also choose the way in which the ranges you provide will be merged with any existing masks. The *append* argument allows you to specify the way in which new mask ranges will interact with any existing masks. By default, these masks will be the "union" of the new mask and any existing masks, but you can also specify that these masks be the mask that results from when you "intersect" the current mask and the new mask, or that the new mask simply "replace" the current mask. The *append* argument can be used in combination with the *invert* argument to make things even more interesting. In this case, the inversion of the mask will happen before it is combined with the existing mask. For simplicity, I will only demonstrate this on *rowmask*, but it also works for *colmask*. Before we begin, lets set the masks back to being NULL again.

```
> ## 1st lets null out the masks so we can have a fresh start.
> colmask(maskTest) <- NULL
> rowmask(maskTest) <- NULL
```

Then we can do a series of examples, starting with the default which uses the "union" value for the *append* argument.

```
> ## Then we can demonstrate how the append argument works
> rowmask(maskTest) <- IRanges(start=1,end=3)
> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                 names
[1] ##########################...######################## Human
[2] ##########################...######################## Chimp
[3] ##########################...######################## Cow
[4] ----------------------AAAA...------------------------ Mouse
[5] --------------------------...------------------------ Rat
[6] --------------------------...------------------------ Dog
[7] --------------CGGCTCCGCAGC...------------------------ Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------ Salmon

> rowmask(maskTest,append="intersect") <- IRanges(start=2,end=5)
> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                 names
[1] -----TCCCGTCTCCGCAGCAAAAAA...TCACAATTAAAAAAAAAAAAAAAA Human
[2] ##########################...######################## Chimp
[3] ##########################...######################## Cow
[4] ----------------------AAAA...------------------------ Mouse
[5] --------------------------...------------------------ Rat
[6] --------------------------...------------------------ Dog
[7] --------------CGGCTCCGCAGC...------------------------ Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------ Salmon

> rowmask(maskTest,append="replace") <- IRanges(start=5,end=8)
> maskTest

DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                 names
[1] -----TCCCGTCTCCGCAGCAAAAAA...TCACAATTAAAAAAAAAAAAAAAA Human
[2] --------------------------...------------------------ Chimp
[3] --------------------------...------------------------ Cow
[4] ----------------------AAAA...------------------------ Mouse
[5] ##########################...######################## Rat
[6] ##########################...######################## Dog
[7] ##########################...######################## Chicken
[8] ##########################...######################## Salmon

> rowmask(maskTest,append="replace",invert=TRUE) <- IRanges(start=5,end=8)
> maskTest
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                    names
[1] ##########################...######################### Human
[2] ##########################...######################### Chimp
[3] ##########################...######################### Cow
[4] ##########################...######################### Mouse
[5] --------------------------...------------------------- Rat
[6] --------------------------...------------------------- Dog
[7] --------------CGGCTCCGCAGC...------------------------ Chicken
[8] GGGGGAGACTTCAGAAGTTGTTGTCC...------------------------ Salmon
```

```
> rowmask(maskTest,append="union") <- IRanges(start=7,end=8)
> maskTest
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                    names
[1] ##########################...######################### Human
[2] ##########################...######################### Chimp
[3] ##########################...######################### Cow
[4] ##########################...######################### Mouse
[5] --------------------------...------------------------- Rat
[6] --------------------------...------------------------- Dog
[7] ##########################...######################### Chicken
[8] ##########################...######################### Salmon
```

The function `maskMotif` works on *MultipleAlignment* objects too, and takes the same arguments that it does elsewhere. `maskMotif` is useful for masking occurances of a string from columns where it is present in the consensus sequence.

```
> tataMasked <- maskMotif(origMAlign, "TATA")
> colmask(tataMasked)
```

```
NormalIRanges object with 5 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]       811       814         4
  [2]      1180      1183         4
  [3]      1186      1191         6
  [4]      1204      1207         4
  [5]      1218      1221         4
```

`maskGaps` also operates on columns and will mask collumns based on the fraction of each column that contains gaps *min.fraction* along with the width of columns that contain this fraction of gaps *min.block.width*.

```
> autoMasked <- maskGaps(origMAlign, min.fraction=0.5, min.block.width=4)
> autoMasked
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                    names
[1] ##########################...######################### Human
[2] ##########################...######################### Chimp
[3] ##########################...######################### Cow
[4] ##########################...######################### Mouse
```

```
[5] ##########################...######################### Rat
[6] ##########################...######################### Dog
[7] ##########################...######################### Chicken
[8] ##########################...######################### Salmon
```

Sometimes you may want to cast your *MultipleAlignment* to be a matrix for usage eslewhere. *as.matrix* is supported for these circumstances. The ability to convert one object into another is not very unusual so why mention it? Because when you cast your object, the masks WILL be considered so that the masked rows and columns will be left out of the matrix object.

```
> full = as.matrix(origMAlign)
> dim(full)

[1]    8 2343

> partial = as.matrix(autoMasked)
> dim(partial)

[1]    8 1143
```

One example of where you might want to use *as.matrix* is when using the `ape` package. For example if you needed to use the *dist.dna* function you would want to use *as.matrix* followed by `as.alignment` and then the `as.DNAbin` to create a `DNAbin` object for the `dist.dna`.

3 Analytic utilities

Once you have masked the sequence, you can then ask questions about the properties of that sequence. For example, you can look at the alphabet frequency of that sequence. The alphabet frequency will only be for the masked sequence.

```
> alphabetFrequency(autoMasked)

       A   C   G   T M R W S Y K V H D B N   - + .
[1,] 260 351 296 218 0 0 0 0 0 0 0 0 0 0 0  18 0 0
[2,] 171 271 231 128 0 0 0 0 0 0 0 0 0 0 3 339 0 0
[3,] 277 360 275 209 0 0 0 0 0 0 0 0 0 0 0  22 0 0
[4,] 265 343 277 226 0 0 0 0 0 0 0 0 0 0 0  32 0 0
[5,] 251 345 287 229 0 0 0 0 0 0 0 0 0 0 0  31 0 0
[6,] 160 285 241 118 0 0 0 0 0 0 0 0 0 0 0 339 0 0
[7,] 224 342 273 190 0 0 0 0 0 0 0 0 0 0 0 114 0 0
[8,] 268 289 273 262 0 0 0 0 0 0 0 0 0 0 0  51 0 0
```

You can also calculate a consensus matrix, extract the consensus string or look at the consensus views. These methods too will all consider the masking when you run them.

```
> consensusMatrix(autoMasked, baseOnly=TRUE)[, 84:90]

      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
A        0    3    0    3    0    3    3
C        3    0    0    1    1    1    2
G        1    2    4    1    2    0    0
T        1    0    1    0    1    1    0
other    3    3    3    3    4    3    3
```

```
> substr(consensusString(autoMasked),80,130)

[1] "####CRGABAMGTCA-YRGCTTCTCYGTSCAWAGGCRRTGRCYTGTTYTCG"

> consensusViews(autoMasked)

  Views on a 2343-letter BString subject
subject: ----------------------VWVMKYYBSRST...---------------------------------
views:
    start  end width
[1]    84  325   242 [CRGABAMGTCA-YRGCTTCTCYGTSCA...WGCGSCTCSGCYGGSGCYRYCCTGSGG]
[2]   330  332     3 [CCR]
[3]   338 1191   854 [CTGCTGCTGYCGGGVCACGGCGYYCGG...CTGMTAGTTTTTATGTATAAATATATA]
[4]  1198 1241    44 [ATAAAATATAAKAC--TTTTTATAYRSCARATGTAAAAATTCAA]
```

You can also cluster the alignments based on their distance to each other. Because you must pass in a DNAStringSet, the clustering will also take into account the masking. So for example, you can see how clustering the unmasked `DNAMultipleAlignment` will draw a funky looking tree.

```
> sdist <- stringDist(as(origMAlign,"DNAStringSet"), method="hamming")
> clust <- hclust(sdist, method = "single")
> pdf(file="badTree.pdf")
> plot(clust)
> dev.off()

null device
          1
```

But, if we use the gap-masked `DNAMultipleAlignment`, to remove the long uninformative regions, and then make our plot, we can see the real relationships.

```
> sdist <- stringDist(as(autoMasked,"DNAStringSet"), method="hamming")
> clust <- hclust(sdist, method = "single")
> pdf(file="goodTree.pdf")
> plot(clust)
> dev.off()

null device
          1

> fourgroups <- cutree(clust, 4)
> fourgroups

  Human   Chimp     Cow   Mouse     Rat     Dog Chicken  Salmon
      1       2       1       1       1       2       3       4
```

In the "good" plot, the Salmon sequence is once again the most distant which is what we expect to see. A closer examination of the sequence reveals that the similarity between the mouse, rat and human sequences was being inflated by virtue of the fact that those sequences were simply much longer (had more information than) the other species represented. This is what caused the "funky" result. The relationship between the sequences in the funky tree was being driven by extra "length" in the rodent/mouse/human sequences, instead of by the similarity of the conserved regions.

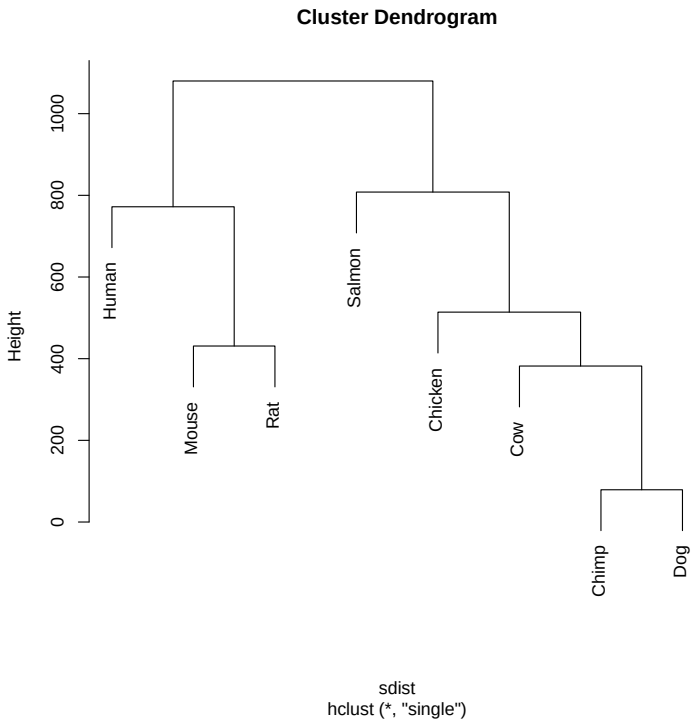

Figure 1: Funky tree produced by using unmasked strings.

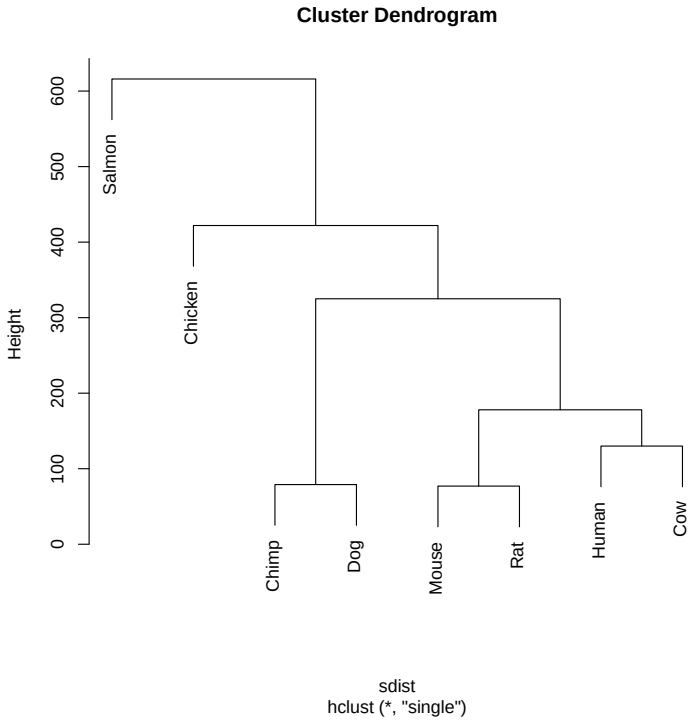

Figure 2: A tree produced by using strings with masked gaps.

4 Exporting to file

One possible export option is to write to fasta files If you need to write your *MultipleAlignment* object out as a fasta file, you can cast it to a *DNAStringSet* and then write it out as a fasta file like so:

```
> DNAStr = as(origMAlign, "DNAStringSet")
> writeXStringSet(DNAStr, file="myFile.fa")
```

One other format that is of interest is the Phylip format. The Phylip format stores the column masking of your object as well as the sequence that you are exporting. So if you have masked the sequence and you write out a Phylip file, this mask will be recorded into the file you export. As with the fasta example above, any rows that you have masked out will be removed from the exported file.

```
> write.phylip(phylipMAlign, filepath="myFile.txt")
```

5 Session Information

All of the output in this vignette was produced under the following conditions:

```
> sessionInfo()
```

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.2 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
[1] Biostrings_2.52.0   XVector_0.24.0      IRanges_2.18.0
[4] S4Vectors_0.22.0    BiocGenerics_0.30.0

loaded via a namespace (and not attached):
[1] zlibbioc_1.30.0 compiler_3.6.0  tools_3.6.0
```