Motif import, export, and manipulation

Benjamin Jean-Marie Tremblay*

17 October 2021

Abstract

The universalmotif package offers a number of functions to manipulate motifs. These are introduced and explored here, including those relating to: import, export, motif modification, creation, visualization, and other miscellaneous utilities.

Contents

1 Introduction 2

2 The universalmotif class and conversion utilities 2
  2.1 The universalmotif class 2
  2.2 Converting to and from another package’s class 4

3 Importing and exporting motifs 6
  3.1 Importing 6
  3.2 Exporting 6

4 Modifying motifs and related functions 6
  4.1 Converting motif type 6
  4.2 Merging motifs 9
  4.3 Motif reverse complement 10
  4.4 Switching between DNA and RNA alphabets 11
  4.5 Motif trimming 12
  4.6 Rounding motifs 13

5 Motif creation 14
  5.1 From a PCM/PPM/PWM/ICM matrix 14
  5.2 From sequences or character strings 15
  5.3 Generating random motifs 15

6 Motif visualization 17
  6.1 Motif logos 17
  6.2 Stacked motif logos 19
  6.3 Plot arbitrary text logos 20

7 Higher-order motifs 20

8 Tidy motif manipulation with the universalmotif_df data structure 23

9 Miscellaneous motif utilities 25
  9.1 DNA/RNA/AA consensus functions 25
  9.2 Filter through lists of motifs 26

*benjamin.tremblay@uwaterloo.ca
1 Introduction

This vignette will introduce the `universalmotif` class and its structure, the import and export of motifs in R, basic motif manipulation, creation, and visualization. For an introduction to sequence motifs, see the introductory vignette. For sequence-related utilities, see the sequences vignette. For motif comparisons and P-values, see the motif comparisons and P-values vignette.

2 The universalmotif class and conversion utilities

2.1 The universalmotif class

The `universalmotif` package stores motifs using the `universalmotif` class. The most basic `universalmotif` object exposes the `name`, `alphabet`, `type`, `strand`, `icscore`, `consensus`, and `motif` slots; furthermore, the `pseudocount` and `bkg` slots are also stored but not shown. `universalmotif` class motifs can be PCM, PPM, PWM, or ICM type.

```r
library(universalmotif)
data(examplemotif)
examplemotif
```

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: PPM
#> Strands: +-;
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: TATAWAW
#>
#> T A T A W A W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5
```

A brief description of all the available slots:

- `name`: motif name
- `altname`: (optional) alternative motif name
- `family`: (optional) a word representing the transcription factor or matrix family
- `organism`: (optional) organism of origin
- `motif`: the actual motif matrix
- `alphabet`: motif alphabet
- `type`: motif ‘type’, one of PCM, PPM, PWM, ICM; see the introductory vignette
- `icscore`: (generated automatically) Sum of information content for the motif
- `nsites`: (optional) number of sites the motif was created from
- **pseudocount**: this value to added to the motif matrix during certain type conversions; this is necessary to avoid -Inf values from appearing in PWM type motifs
- **bkg**: a named vector of probabilities which represent the background letter frequencies
- **bkgsites**: (optional) total number of background sequences from motif creation
- **consensus**: (generated automatically) for DNA/RNA/AA motifs, the motif consensus
- **strand**: strand motif can be found on
- **pval**: (optional) P-value from de novo motif search
- **qval**: (optional) Q-value from de novo motif search
- **eval**: (optional) E-value from de novo motif search
- **multifreq**: (optional) higher-order motif representations.
- **extrainfo**: (optional) any extra motif information that cannot fit in the existing slots

The other slots will be shown as they are filled.

```r
library(universalmotif)
data(examplemotif)

## The various slots can be accessed individually using `[]`
examplemotif["consensus"]

## To change a slot, use `<-`
examplemotif["family"] <- "My motif family"
```

Though the slots can easily be changed manually with `<-`, a number of safeguards have been put in place for some of the slots which will prevent incorrect values from being introduced.

```r
library(universalmotif)
data(examplemotif)

## The consensus slot is dependent on the motif matrix
examplemotif["consensus"]

## Changing this would mean it no longer matches the motif
examplemotif["consensus"] <- "GGGAGAG"
```
```r
#> Error in .local(x, i, ..., value): this slot is unmodifiable with [<-

## Another example of trying to change a protected slot:

```examplemotif["strand"]` <- "x"
#> Error in eval(expr, envir, enclos): * strand must be one of +, -, +-```

Below the exposed metadata slots, the actual ‘motif’ matrix is shown. Each position is its own column: row names showing the alphabet letters, and the column names showing the consensus letter at each position.

### 2.2 Converting to and from another package’s class

The `universalmotif` package aims to unify most of the motif-related Bioconductor packages by providing the `convert_motifs()` function. This allows for easy transition between supported packages (see `?convert_motifs` for a complete list of supported packages). Should you ever come across a motif class from another Bioconductor package which is not supported by the `universalmotif` package, but believe it should be, then feel free to bring it up with me.

The `convert_motifs` function is embedded in most of the `universalmotif` functions, meaning that compatible motif classes from other packages can be used without needed to manually convert them first. However keep in mind some conversions are final. Furthermore, internally, all motifs regardless of class are handled as `universalmotif` objects, even if the returning class is not. This will result in at times slightly different objects (though usually no information should be lost).

```r
library(universalmotif)
library(MotifDb)
data(examplemotif)
data(MA0003.2)

# convert from a `universalmotif` motif to another class

convert_motifs(examplemotif, "TFBSTools-PWMatrix")
#> Note: motif [motif] has an empty nsites slot, using 100.
#> An object of class PWMatrix
#> ID:
#> Name: motif
#> Matrix Class: Unknown
#> strand: *
#> Pseudocounts: 1
#> Tags:
#> list()
#> Background:
#> A C G T
#> 0.25 0.25 0.25 0.25
#> Matrix:
#> T A T A W A W
#> A -6.658211 1.989247 -6.658211 1.989247 0.9928402 1.989247 0.9928402
#> C -6.658211 -6.658211 -6.658211 -6.658211 -6.6582115 -6.658211 -6.6582115
#> G -6.658211 -6.658211 -6.658211 -6.658211 -6.6582115 -6.658211 -6.6582115
#> T 1.989247 -6.658211 1.989247 -6.658211 0.9928402 -6.658211 0.9928402

# convert to universalmotif

convert_motifs(MA0003.2)
#>```

```
Motif name: TFAP2A
Alternate name: MA0003.2
Family: Helix-Loop-Helix
Organism: 9606
Alphabet: DNA
Type: PCM
Strands: +
Total IC: 12.9
Pseudocount: 1
Consensus: NNNGCCYSGGCA
Target sites: 5098
Extra info: [centrality_logp] -4343

```r
convert_motifs(MotifDb[1], "TFBSTools-ICMatrix")
```

Note: motif [ABF2] has an empty nsites slot, using 100.

An object of class ICMatrix
ID: badis.ABF2
Name: ABF2
Matrix Class: Unknown
strand: *
Pseudocounts: 1
Schneider correction: FALSE
Tags:
$dataSource
[1] "ScerTF"

Background:
A C G T
0.25 0.25 0.25 0.25
Matrix:
```
   T  C  T  A  G  A
A 0.08997357 0.02119039 0.02119039 1.64861232 0.02119039 1.43716039
C 0.08997357 1.64861232 0.02119039 0.02119039 0.02119039 0.02119039 0.03430887
G 0.02188546 0.02119039 0.02119039 0.02119039 1.64861232 0.02119039 0.03430887
T 0.78058151 0.02119039 1.64861232 0.02119039 0.02119039 0.02119039 0.03430887
```
3 Importing and exporting motifs

3.1 Importing

The universalmotif package offers a number of read_*() functions to allow for easy import of various motif formats. These include:

- **read_cisbp()**: CIS-BP (Weirauch et al. 2014)
- **read_homer()**: HOMER (Heinz et al. 2010)
- **read_jaspar()**: JASPAR (Khan et al. 2018)
- **read_matrix()**: generic reader for simply formatted motifs
- **read_meme()**: MEME (Bailey et al. 2009)
- **read_motifs()**: native universalmotif format (not recommended; use saveRDS() instead)
- **read_transfac()**: TRANSFAC (Wingender et al. 1996)
- **read_uniprobe()**: UniPROBE (Hume et al. 2015)

These functions should work natively with these formats, but if you are generating your own motifs in one of these formats than it must adhere quite strictly to the format. An example of each of these is included in this package (see system.file("extdata", package="universalmotif")). If you know of additional motif formats which are not supported in the universalmotif package that you believe should be, or of any mistakes in the way the universalmotif package parses supported formats, then please let me know.

3.2 Exporting

Compatible motif classes can be written to disk using:

- **write_homer()**
- **write_jaspar()**
- **write_matrix()**
- **write_meme()**
- **write_motifs()**
- **write_transfac()**

The write_matrix() function, similar to its read_matrix() counterpart, can write motifs as simple matrices with an optional header. Additionally, please keep in mind format limitations. For example, multiple MEME motifs written to a single file will all share the same alphabet, with identical background letter frequencies.

4 Modifying motifs and related functions

4.1 Converting motif type

Any universalmotif object can transition between PCM, PPM, PWM, and ICM types seamlessly using the convert_type() function. The only exception to this is if the ICM calculation is performed with sample correction, or as relative entropy. If this occurs, then back conversion to another type will be inaccurate (and convert_type() would not warn you, since it won’t know this has taken place).

```r
library(universalmotif)
data(examplemotif)

## This motif is currently a PPM:

examplemotif["type"]
#> [1] "PPM"
```

When converting to PCM, the nsites slot is needed to tell it how many sequences it originated from. If empty, 100 is used.
for converting to PWM, the pseudocount slot is used to determine if any correction should be applied:

```r
examplemotif[['pseudocount']]
```
There are a couple of additional options for ICM conversion: `nsize_correction` and `relative_entropy`. The former uses the `TFBSTools:::schneider_correction()` function (and thus requires that the `TFBSTools` package be installed) for sample size correction. The latter uses the `bkg` slot to calculate information content. See the `IntroductionToSequenceMotifs` vignette for an overview on the various types of ICM calculations.

```r
examplemotif["nsites"] <- 10
convert_type(examplemotif, "ICM", nsize_correction = FALSE)
```

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: ICM
#> Strands: +-  
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: TATAWAW
#> Target sites: 10

#> T A T A W A W
#> A 0 2 0 2 0.5 2 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 2 0 2 0.5 0 0.5
```

```r
convert_type(examplemotif, "ICM", nsize_correction = TRUE)
```

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: ICM
#> Strands: +-  
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: TATAWAW
#> Target sites: 10

#> T A T A W A W
#> A 0.00 1.75 0.00 1.75 0.38 1.75 0.38
#> C 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> G 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> T 1.75 0.00 1.75 0.00 0.38 0.00 0.38
```

```r
examplemotif["bkg"] <- c(A = 0.4, C = 0.1, G = 0.1, T = 0.4)
convert_type(examplemotif, "ICM", relative_entropy = TRUE)
```

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: ICM
#> Strands: +-  
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: TATAWAW
#> Target sites: 10

#> T A T A W A W
#> A 0.00 1.75 0.00 1.75 0.38 1.75 0.38
#> C 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> G 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> T 1.75 0.00 1.75 0.00 0.38 0.00 0.38
```
4.2 Merging motifs

The universalmotif package includes the `merge_motifs()` function to combine motifs. Motifs are first aligned, and the best match found before the motif matrices are averaged. The implementation for this is identical to that used by `compare_motifs()` (see the motif comparisons vignette for more information).

```r
library(universalmotif)

m1 <- create_motif("TTAAACCCC", name = "1")
m2 <- create_motif("AACC", name = "2")
m3 <- create_motif("AACCCCGG", name = "3")

view_motifs(c(m1, m2, m3),
            show.positions.once = FALSE, show.names = FALSE)
```

```r
view_motifs(merge_motifs(c(m1, m2, m3), method = "PCC"))
```
This functionality can also be automated to reduce the number of overly similar motifs in larger datasets via the `merge_similar()` function.

```r
library(universalmotif)
library(MotifDb)

motifs <- filter_motifs(MotifDb, family = "bHLH")[1:100]
#> motifs converted to class 'universalmotif'
length(motifs)
#> [1] 100

motifs <- merge_similar(motifs)
length(motifs)
#> [1] 63
```

Comparison and merging parameters can be fine-tuned as users wish. See the `compare_motifs()` and `merge_motifs()` documentation for more details, as well as the “Motif comparison and P-values” vignette.

### 4.3 Motif reverse complement

Get the reverse complement of a motif.

```r
library(universalmotif)
data(examplemotif)

## Quickly switch to the reverse complement of a motif

## Original:

#> Motif name:  motif
#> Alphabet:  DNA
#> Type:  PPM
#> Strands:  +-  
#> Total IC:  11.54
#> Pseudocount:  1
#> Consensus:  TATAWAW

#> T   A   T   A   W   A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5
```
4.4 Switching between DNA and RNA alphabets

Since not all motif formats or programs support RNA alphabets by default, the `switch_alph()` function can quickly go between DNA and RNA motifs.

```r
library(universalmotif)
data(examplemotif)

## DNA --> RNA

switch_alph(examplemotif)
```

```r
#> Motif name: motif
#> Alphabet: RNA
#> Type: PPM
#> Strands: +-  
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: UAUAWAW
#>
#> U A U A W A W
#> A 0 0 1 0 1 0 0 0.
#> C 0 0 0 0 0 0 0 0 0
#> G 0 0 0 0 0 0 0 0 0
#> U 0 1 0 1 0 0 0 0 0
```

## RNA --> DNA

```r
motif <- create_motif(alphabet = "RNA")
motif
```

```r
#> Motif name: motif
#> Alphabet: RNA
#> Type: PPM
#> Strands: +-  
```

```r
# W T W T A T A
# A 0.5 0 0.5 0 1 0 1
# C 0 0 0 0 0 0 0 0
# G 0 0 0 0 0 0 0 0
# T 0.5 1 0.5 1 0 1 0
```
4.5 Motif trimming

Get rid of low information content edges on motifs, such as NNCGGGCNN to CGGGC. The ‘amount’ of trimming can also be controlled by setting a minimum required information content, as well as the direction of trimming (by default both edges are trimmed).

```r
library(universalmotif)

motif <- create_motif("NNGCSGCGGNN")
```

```r
trim_motifs(motif)
```

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: PPM
#> Strands: +- 
#> Total IC: 13
#> Pseudocount: 0
#> Consensus: NNGCSGCGGNN
```

```r
#> N N G C S G C G G N N
#> A 0.25 0.25 0 0 0.0 0 0 0 0 0.25 0.25
#> C 0.25 0.25 0 1 0.5 0 1 0 0 0.25 0.25
#> G 0.25 0.25 1 0 0.5 1 0 1 1 0.25 0.25
#> T 0.25 0.25 0 0 0.0 0 0 0 0 0.25 0.25
```
4.6 Rounding motifs

Round off near-zero probabilities.

```r
motif1 <- create_motif("ATCGATGC", pseudocount = 10, type = "PPM", nsites = 100)
motif2 <- round_motif(motif1)
view_motifs(c(motif1, motif2))
```
5 Motif creation

Though universalmotif class motifs can be created using the `new` constructor, the universalmotif package provides the `create_motif()` function which aims to provide a simpler interface to motif creation. The universalmotif class was initially designed to work natively with DNA, RNA, and amino acid motifs. Currently though, it can handle any custom alphabet just as easily. The only downsides to custom alphabets is the lack of support for certain slots such as the `consensus` and `strand` slots.

The `create_motif()` function will be introduced here only briefly; see `?create_motif` for details.

5.1 From a PCM/PPM/PWM/ICM matrix

Should you wish to make use of the universalmotif functions starting from a motif class unsupported by `convert_motifs()`, you can instead manually create universalmotif class motifs using the `create_motif()` function and the motif matrix.

```r
motif.matrix <- matrix(c(0.7, 0.1, 0.1, 0.1, 0.7, 0.1, 0.1, 0.1, 0.1, 0.7, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.7), nrow = 4)

motif <- create_motif(motif.matrix, alphabet = "RNA", name = "My motif", pseudocount = 1, nsites = 20, strand = "+")

## The 'type', 'icscore' and 'consensus' slots will be filled for you
```

```r
motif
#>
#> Motif name: My motif
#> Alphabet: RNA
```
As a brief aside: if you have a motif formatted simply as a matrix, you can still use it with the universalmotif package functions natively without creating a motif with create_motif(), as convert_motifs() also has the ability to handle motifs formatted simply as matrices. However it is much safer to first format the motif beforehand with create_motif().

5.2 From sequences or character strings

If all you have is a particular consensus sequence in mind, you can easily create a full motif using create_motif(). This can be convenient if you’d like to create a quick motif to use with an external program such as from the MEME suite or HOMER. Note that ambiguity letters can be used with single strings.

```r
motif <- create_motif("CCNSNGG", nsites = 50, pseudocount = 1)
## Now to disk:
## write_meme(motif, "meme_motif.txt")
```

motif

```r
#> Motif name: motif
#> Alphabet: DNA
#> Type: PPM
#> Strands: +-  
#> Total IC: 8.39
#> Pseudocount: 1
#> Consensus: C C N S N G G
#> A 0.00 0.00 0.22 0.0 0.22 0.00 0.00
#> C 0.99 0.99 0.26 0.5 0.26 0.00 0.00
#> G 0.00 0.00 0.26 0.5 0.26 0.99 0.99
#> T 0.00 0.00 0.26 0.0 0.26 0.00 0.00
```

5.3 Generating random motifs

If you wish to, it’s easy to create random motifs. The values within the motif are generated using rgamma() to avoid creating low information content motifs. If background probabilities are not provided, then they are generated with rpois().

```r
create_motif()
#>
#> Motif name: motif
```

15
You can change the probabilities used to generate the values within the motif matrix:

```
create_motif(bkg = c(A = 0.2, C = 0.4, G = 0.2, T = 0.2))
```

With a custom alphabet:

```
create_motif(alphabet = "QWERTY")
```
6 Motif visualization

6.1 Motif logos

There are several packages which offer motif visualization capabilities, such as seqLogo, motifStack, and ggseqlogo. The universalmotif package has its own implementation via the function view_motifs(), which renders motifs using the ggplot2 package (similar to ggseqlogo). Here I will briefly show how to use these to visualize universalmotif class motifs.

```r
library(universalmotif)
data(examplemotif)

## With the native `view_motifs` function:
view_motifs(examplemotif)
```

The view_motifs() function generates ggplot objects; feel free to manipulate them as such. For example, flipping the position numbers for larger motifs (where the text spacing can become tight):

```r
view_motifs(create_motif(15)) +
ggplot2::theme(
    axis.text.x = ggplob2::element_text(angle = 90, hjust = 1)
)
```

A large number of options are available for tuning the way motifs are plotted in view_motifs(). Visit the documentation for more information.

Using the other Bioconductor packages to view universalmotif motifs is fairly easy as well:

```r
## For all the following examples, simply passing the functions a PPM is
## sufficient
motif <- convert_type(examplemotif, "PPM")
```
## Only need the matrix itself
motif <- motif["motif"]

## seqLogo:
seqLogo::seqLogo(motif)

## motifStack:
motifStack::plotMotifLogo(motif)

## ggseqlogo:
ggseqlogo::ggseqlogo(motif)

#> Loading required namespace: Cairo

#> Warning: The `scale` argument of `guides()` cannot be `FALSE`. Use "none" instead as of ggplot2 3.3.4.
#> i The deprecated feature was likely used in the ggseqlogo package.
#> Please report the issue at <https://github.com/omarwagih/ggseqlogo/issues>.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
6.2 Stacked motif logos

The motifStack package allows for a number of different motif stacking visualizations. The universalmotif package, while not capable of emulating most of these, still offers basic stacking via `view_motifs()`. The motifs are aligned using `compare_motifs()`.

```r
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[50:54])
view_motifs(motifs, show.positions.once = FALSE, names.pos = "right")
```
6.3 Plot arbitrary text logos

The logo plotting capabilities of `view_motifs()` can be used for any kind of arbitrary text logo. All you need is a numeric matrix (the heights of the characters), with the desired characters as row names. The following example is taken from the `view_logo()` documentation.

```r
library(universalmotif)
data(examplemotif)

## Start from a numeric matrix:
toplot <- examplemotif["motif"]

# Adjust the character heights as you wish (negative values are possible):
toplot[20] <- -0.5

# Mix and match the number of characters per letter/position:
rownames(toplot)[1] <- "AA"
toplot <- toplot[c(1, 4), ]

toplot
#>   T A T A W A W
#> AA 0 1 0 1 0.5 1 0.5
#> T 2 0 1 0 -0.5 0 0.5

view_logo(toplot)
```

7 Higher-order motifs

Though PCM, PPM, PWM, and ICM type motifs are still widely used today, a few ‘next generation’ motif formats have been proposed. These wish to add another layer of information to motifs: positional interdependence. To illustrate this, consider the following sequences:

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CAAAACC</td>
</tr>
<tr>
<td>2</td>
<td>CAAAACC</td>
</tr>
</tbody>
</table>

Table 1: Example sequences.
### Table 2: Position Probability Matrix.

<table>
<thead>
<tr>
<th>Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>C</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>G</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>T</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Based on the PPM representation, all three of CAAAACC, CTTTTCC, and CTATACC are equally likely. Though looking at the starting sequences, should CTATACC really be considered so? For transcription factor binding sites, this sometimes is not the case. By incorporating this type of information into the motif, it can allow for increased accuracy in motif searching. A few example implementations of this include: TFFM by Mathelier and Wasserman (2013), BaMM by Siebert and Soding (2016), and KSM by Guo et al. (2018).

The universalmotif package implements its own, rather simplified, version of this concept. Plainly, the standard PPM has been extended to include \( k \)-letter frequencies, with \( k \) being any number higher than 1. For example, the 2-letter version of the table 2 motif would be:

### Table 3: 2-letter probability matrix.

<table>
<thead>
<tr>
<th>Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>AC</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>AG</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>AT</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CA</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CC</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>CG</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CT</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>GA</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>GC</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>GG</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>GT</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>TA</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>TC</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>TG</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>TT</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

This format shows the probability of each letter combined with the probability of the letter in the next position. The seventh column has been dropped, since it is not needed: the information in the sixth column is sufficient, and there is no eighth position to draw 2-letter probabilities from. Now, the probability of getting CTATACC is no longer equal to CTTTTCC and CAAAACC. This information is kept in the multifreq slot of universalmotif class motifs. To add this information, use the add_multifreq() function.
library(universalmotif)

motif <- create_motif("CWWWWCC", nsites = 6)
sequences <- DNAStringSet(rep(c("CAAAACC", "CTTTTCC"), 3))
motif.k2 <- add_multifreq(motif, sequences, add.k = 2)

## Alternatively:
# motif.k2 <- create_motif(sequences, add.multifreq = 2)

motif.k2

view_motifs(motif.k2, use.freq = 2)

To plot these motifs, use `view_motifs()`:

This information is most useful with functions such as `scan_sequences()` and `enrich_motifs()`. Though other tools in the `universalmotif` can work with `multifreq` motifs (such as `motif_pvalue()`, `compare_motifs()`), keep in mind they are not as well supported as regular motifs (getting P-values from `multifreq` motifs is exponentially slower, and P-values from using `compare_motifs()` for `multifreq` motifs are not available by default). See the sequences vignette for using `scan_sequences()` with the `multifreq` slot.
8 Tidy motif manipulation with the universalmotif_df data structure

For those who enjoy using the tidyverse functions for data handling, motifs can additionally be represented as the modified data.frame format: universalmotif_df. This format allows one to modify motif slots for multiple motifs simultaneously using the universalmotif_df columns, and then return to a list of motifs afterwards to resume use with universalmotif package functions. A few key functions have been provided in relation to this format:

- **to_df()**: Generate a universalmotif_df object from a list of motifs.
- **update_motifs()**: After modifying the universalmotif_df object, apply these modifications to the actual universalmotif objects (contained within the motif column).
- **to_list()**: Return to a list of universalmotif objects for use with universalmotif package functions. Note that it is not required to use update_motifs() before using to_list(), as modifications will be checked for and applied if found.
- **requires_update()**: Boolean check as to whether the universalmotif objects and the universalmotif_df columns differ and require either a update_motifs() or to_list() call to re-sync them.

library(universalmotif)
library(MotifDb)

## Obtain a `universalmotif_df` object

motifs <- to_df(MotifDb)

head(motifs)

```r
#> #> motif name altname organism consensus alphabet strand icscore
#> #> 1 <mot:ABF2> ABF2 badis.ABF2 Scerevisiae TCTAGA DNA +- 9.371235
#> 2 <mot:CAT8> CAT8 badis.CAT8 Scerevisiae CCGGAN DNA +- 7.538740
#> 3 <mot:CST6> CST6 badis.CST6 Scerevisiae TGACGT DNA +- 9.801864
#> 4 <mot:ECM23> ECM23 badis.ECM23 Scerevisiae AGATC DNA +- 6.567494
#> 5 <mot:EDS1> EDS1 badis.EDS1 Scerevisiae GGAANAA DNA +- 9.314287
#> 6 <mot:FKH2> FKH2 badis.FKH2 Scerevisiae GTAAACA DNA +- 11.525400
#> type pseudocount bkg dataSource
#> 1 PPM 1 0.25, 0..... ScerTF
#> 2 PPM 1 0.25, 0..... ScerTF
#> 3 PPM 1 0.25, 0..... ScerTF
#> 4 PPM 1 0.25, 0..... ScerTF
#> 5 PPM 1 0.25, 0..... ScerTF
#> 6 PPM 1 0.25, 0..... ScerTF
#>
#> [Hidden empty columns: family, nsites, bkgsites, pval, qval, eval.]
```

Some tidy manipulation:

```r
library(dplyr)

motifs <- motifs %>%
  mutate(bkg = case_when(
    organism == "Athaliana" ~ list(c(A = 0.32, C = 0.18, G = 0.18, T = 0.32)),
    TRUE ~ list(c(A = 0.25, C = 0.25, G = 0.25, T = 0.25))
  )

head(filter(motifs, organism == "Athaliana"))

#> #> motif name altname family organism consensus alphabet
#> #> 1 * <mot:ORA59> ORA59 M0005_1.02 AP2 Athaliana MGCCGCGC DNA
#> 2 * <mot:WIN1> WIN1 M0006_1.02 AP2 Athaliana NCRCCGCGNNN DNA
```
Feel free to add columns as well. You can add 1d vectors which will be added to the `extrainfo` slots of motifs. (Note that they will be coerced to character vectors!)

```r
motifs <- motifs %>%
  mutate(MotifIndex = 1:n())
```

```r
to_list(motifs)[[1]]
```

```r
# Motif name: ABF2
# Alternate name: badis.ABF2
# Organism: Scerevisiae
# Alphabet: DNA
# Type: PPM
# Strands: +- 
# Total IC: 9.37 
# Pseudocount: 1
# Consensus: TCTAGA
# Extra info: [dataSource] ScerTF
# [MotifIndex] 1
If during the course of your manipulation you’ve generated temporary columns which you wish to drop, you can set `extrainfo = FALSE` to discard all extra columns. Be careful though, this will discard any previously existing `extrainfo` data as well.

```r
> to_list(motifs, extrainfo = FALSE)[[1]]
```

### 9 Miscellaneous motif utilities

A number of convenience functions are included for manipulating motifs.

#### 9.1 DNA/RNA/AA consensus functions

For DNA, RNA and AA motifs, the `universalmotif` will automatically generate a `consensus` string slot. Furthermore, `create_motif()` can generate motifs from consensus strings. The internal functions for these have been made available:

- `consensus_to_ppm()`
- `consensus_to_ppmAA()`
- `get_consensus()`
- `get_consensusAA()`

```r
collection <- list()

library(universalmotif)

get_consensus(c(A = 0.7, C = 0.1, G = 0.1, T = 0.1))
#> [1] "A"

consensus_to_ppm("G")
#> [1] 0.001 0.001 0.997 0.001
```
9.2 Filter through lists of motifs

Filter a list of motifs, using the `universalmotif` slots with `filter_motifs()`.

```r
library(universalmotif)
library(MotifDb)

## Let us extract all of the Arabidopsis and C. elegans motifs
motifs <- filter_motifs(MotifDb, organism = c("Athaliana", "Celegans"))
## Only keeping motifs with sufficient information content and length:
motifs <- filter_motifs(motifs, icscore = 10, width = 10)

head(summarise_motifs(motifs))
```

9.3 Generate random motif matches

Get a random set of sequences which are created using the probabilities of the motif matrix, in effect generating motif sites, with `sample_sites()`.

```r
library(universalmotif)
data(examplemotif)

sample_sites(examplemotif)
```
9.4 Motif shuffling

Shuffle a set of motifs with `shuffle_motifs()`. The original shuffling implementation is taken from the linear shuffling method of `shuffle_sequences()`, described in the sequences vignette.

```r
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[1:50])
head(summarise_motifs(motifs))
#>       name   altname organism consensus alphabet strand  icscore
#> 1  ABF2  badis.ABF2 Scerevisiae   TCTAGA    DNA   +- 9.371235
#> 2 CAT8  badis.CAT8 Scerevisiae   CCGGAN    DNA   +- 7.538740
#> 3 CST6  badis.CST6 Scerevisiae   TGACGT    DNA   +- 8.01864
#> 4 ECM23 badis.ECM23 Scerevisiae   AGATC     DNA   +- 6.567494
#> 5 EDS1  badis.EDS1 Scerevisiae   GGAANAA   DNA   +- 9.314287
#> 6 FKH2  badis.FKH2 Scerevisiae     GTAAA   DNA   +- 11.52540

motifs.shuffled <- shuffle_motifs(motifs, k = 3)
head(summarise_motifs(motifs.shuffled))
#>       name consensus alphabet strand  icscore
#> 1  ABF2 [shuffled]  TRWKCC    DNA   +- 5.856591
#> 2 CAT8 [shuffled]  GGGMAA    DNA   +- 7.261442
#> 3 CST6 [shuffled]  SCSNCA    DNA   +- 5.214041
#> 4 ECM23 [shuffled]  GCRSS    DNA   +- 5.597233
#> 5 EDS1 [shuffled]  AATGAGG   DNA   +- 10.638115
#> 6 FKH2 [shuffled]  GBCMTGT   DNA   +- 8.392421
```

9.5 Scoring and match functions

Motif matches in a set of sequences are typically obtained using logodds scores. Several functions are exposed to reveal some of the internal work that goes on.

- `get_matches()`: show all possible sequence matches above a certain score
- `get_scores()`: obtain all possible scores from all possible sequence matches
- `motif_score()`: translate score thresholds to logodds scores
- `prob_match()`: return probabilities for sequence matches
- `score_match()`: return logodds scores for sequence matches

```r
library(universalmotif)
data(examplemotif)
examplemotif
#>
#> Motif name: motif
#> Alphabet: DNA
#> Type: PPM
#> Strands: +-  
#> Total IC: 11.54
#> Pseudocount: 1
#> Consensus: TATAWAW
#>
#> T A T A W A W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5
```
## Get the min and max possible scores:

```r
motif_score(examplemotif)
#> 0%  100%
#> -46.606 11.929
```

## Show matches above a score of 10:

```r
get_matches(examplemotif, 10)
#> [1] "TATAAAA" "TATATAA" "TATAAAT" "TATATAT"
```

## Get the probability of a match:

```r
prob_match(examplemotif, "TTTTTTT", allow.zero = FALSE)
#> [1] 6.103516e-05
```

## Score a specific sequence:

```r
score_match(examplemotif, "TTTTTTT")
#> [1] -14.012
```

## Take a look at the distribution of scores:

```r
plot(density(get_scores(examplemotif), bw = 5))
```

9.6 Type conversion functions

While `convert_type()` will take care of switching the current type for `universalmotif` objects, the individual type conversion functions are also available for personal use. These are:

- `icm_to_ppm()`
- `pcm_to_ppm()`
- `ppm_to_icm()`
- `ppm_to_pcm()`

N = 16384   Bandwidth = 5
• ppm_to_pwm()
• pwm_to_ppm()

These functions take a one dimensional vector. To use these for matrices:

```
library(universalmotif)

m <- create_motif(type = "PCM")[["motif"]]
m
#> T K G T R A W Y T C
#> A 0 1 0 0 66 78 65 4 0 7
#> C 2 1 14 0 3 22 8 53 0 86
#> G 8 32 86 7 29 0 0 4 0 7
#> T 90 66 0 93 2 0 27 39 100 0

apply(m, 2, pcm_to_ppm)
#> T K G T R A W Y T C
#> [1,] 0.00 0.01 0.00 0.00 0.66 0.78 0.65 0.04 0 0.07
#> [2,] 0.02 0.01 0.14 0.00 0.03 0.22 0.08 0.53 0 0.86
#> [3,] 0.08 0.32 0.86 0.07 0.29 0.00 0.00 0.04 0 0.07
#> [4,] 0.90 0.66 0.00 0.93 0.02 0.00 0.27 0.39 1 0.00
```

Additionally, the `position_ic_score()` can be used to get the total information content per position:

```
library(universalmotif)

position_ic_score(c(0.7, 0.1, 0.1, 0.1))
#> [1] 0.6307803
```

### Session info

```
#> R version 4.3.1 (2023-06-16)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 22.04.3 LTS
#>
#> Matrix products: default
#> BLAS: /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
#>
#> locale:
#> [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
#> [2] LC_TIME=en_GB              LC_CTYPE=C
#> [4] LC_MONETARY=en_US.UTF-8    LC_CTYPE=en_US.UTF-8
#> [5] LC_PAPER=en_US.UTF-8       LC_COLLATE=C
#> [6] LC_ADDRESS=C               LC_CTYPE=C
#> [7] LC_TELEPHONE=C
#> [8] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> time zone: America/New_York
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats4  stats  graphics grDevices utils  datasets methods
#> [8] base
#>
#> other attached packages:
```
References


