Package ‘universalmotif’

May 30, 2024

Title Import, Modify, and Export Motifs with R

Version 1.22.0

URL https://bioconductor.org/packages/universalmotif/

BugReports https://github.com/bjmt/universalmotif/issues

Description Allows for importing most common motif types into R for use by functions provided by other Bioconductor motif-related packages. Motifs can be exported into most major motif formats from various classes as defined by other Bioconductor packages. A suite of motif and sequence manipulation and analysis functions are included, including enrichment, comparison, P-value calculation, shuffling, trimming, higher-order motifs, and others.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

Imports methods, stats, utils, MASS, ggplot2, yaml, IRanges, Rcpp, Biostrings, BiocGenerics, S4Vectors, rlang, grid, MatrixGenerics

Suggests spelling, knitr, bookdown, TFBSTools, rmarkdown, MotifDb, testthat, BiocParallel, seqLogo, motifStack, dplyr, ape, ggtree, processx, ggseqlogo, cowplot, GenomicRanges, ggbio

Enhances PWMEnrich, rGADEM

LinkingTo Rcpp, RcppThread

VignetteBuilder knitr

biocViews MotifAnnotation, MotifDiscovery, DataImport, GeneRegulation

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE, old_usage = TRUE)

Language en-GB

Collate 'RcppExports.R' 'add_multifreq.R' 'compare_motifs.R'
 'universalmotif-class.R' 'convert_motifs.R' 'convert_type.R'
 'create_motif.R' 'create_sequences.R' 'data.R'
 'enrich_motifs.R' 'filter_motifs.R' 'get_bkg.R'

1
'make_DBscores.R' 'merge_motifs.R' 'merge_similar.R'
'motif_clusters.R' 'motif_finder.R' 'motif_peaks.R'
'motif_pvalue.R' 'motif_rc.R' 'motif_tree.R' 'read_cisbp.R'
'read_homer.R' 'read_jaspar.R' 'read_matrix.R' 'read_meme.R'
'read_motifs.R' 'read_transfac.R' 'read_uniprobe.R'
'run_meme.R' 'sample_sites.R' 'scan_sequences.R'
'sequence_complexity.R' 'shuffle_motifs.R'
'shuffle_sequences.R' 'switch_alph.R' 'trim_motifs.R'
'universalmotif-methods.R' 'universalmotif.R'
'universalmotif_df.R' 'utils-internal.R' 'utils-motif.R'
'utils-sequence.R' 'view_logo.R' 'view_motifs.R'
'write_homer.R' 'write_jaspar.R' 'write_matrix.R'
'write_meme.R' 'write_motifs.R' 'write_transfac.R' 'zzz.R'

git_url https://git.bioconductor.org/packages/universalmotif

git_branch RELEASE_3_19

git_last_commit e6c220a

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-29

Author Benjamin Jean-Marie Tremblay [aut, cre]
(https://orcid.org/0000-0002-7441-2951),
Spencer Nystrom [ctb] (https://orcid.org/0000-0003-1000-1579)

Maintainer Benjamin Jean-Marie Tremblay <benjamin.tremblay@uwaterloo.ca>

Contents

add_multifreq .................................................................. 3
ArabidopsisMotif ................................................................. 5
ArabidopsisPromoters ......................................................... 6
compare_motifs ................................................................. 6
convert_motifs ................................................................. 10
convert_type ................................................................. 14
create_motif ................................................................. 16
create_sequences ............................................................ 21
enrich_motifs ................................................................. 22
examplemotif ................................................................. 25
examplemotif2 ............................................................... 26
filter_motifs ................................................................. 26
fontDFroboto ................................................................. 27
get_bkg ......................................................................... 28
JASPAR2018_CORE_DBSCORES ........................................... 30
make_DBscores ............................................................... 30
merge_motifs ................................................................. 32
merge_similar ................................................................. 34
motif_peaks ................................................................. 36
add_multifreq

Description

If the original sequences are available for a particular motif, then they can be used to generate higher-order PPM matrices. See the “Motif import, export, and manipulation” vignette for more information.
Usage

```r
add_multifreq(motif, sequences, add.k = 2:3, RC = FALSE,
             threshold = 0.001, threshold.type = "pvalue", motifs.perseq = 1,
             add.bkg = FALSE)
```

Arguments

- **motif**: See `convert_motifs()` for acceptable formats. If the motif is not a `universalmotif` motif, then it will be converted.
- **sequences**: `XStringSet` The alphabet must match that of the motif. If these sequences are all the same length as the motif, then they are all used to generate the multi-freq matrices. Otherwise `scan_sequences()` is first run to find the best sequence stretches within these.
- **add.k**: numeric(1) The k-let lengths to add.
- **RC**: logical(1) If TRUE, check reverse complement of the input sequences. Only available for DNA/RNA.
- **threshold**: numeric(1) See details.
- **threshold.type**: character(1) One of `c('pvalue', 'qvalue', 'logodds', 'logodds.abs')`. See details.
- **motifs.perseq**: numeric(1) If `scan_sequences()` is run, then this indicates how many hits from each sequence is to be used.
- **add.bkg**: logical(1) Indicate whether to add corresponding higher order background information to the motif. Can sometimes be detrimental when the input consists of few short sequences, which can increase the likelihood of adding zero or near-zero probabilities.

Details

See `scan_sequences()` for more info on scanning parameters.

At each position in the motif, then the probability of each k-let covering from the initial position to ncol - 1 is calculated. Only positions within the motif are considered: this means that the final k-let probability matrix will have ncol - 1 fewer columns. Calculating k-let probabilities for the missing columns would be trivial however, as you would only need the background frequencies. Since these would not be useful for `scan_sequences()` though, they are not calculated.

Currently `add_multifreq()` does not try to stay faithful to the default motif matrix when generating multifreq matrices. This means that if the sequences used for training are completely different from the actual motif, the multifreq matrices will be as well. However this is only really a problem if you supply `add_multifreq()` with a set of sequences of the same length as the motif. In this case `add_multifreq()` is forced to create the multifreq matrices from these sequences. Otherwise `add_multifreq()` will scan the input sequences for the motif and use the best matches to construct the multifreq matrices.

This ‘multifreq’ representation is only really useful within the `universalmotif` environment. Despite this, if you wish it can be preserved in text using `write_motifs()`.
A note on motif size:
The number of rows for each k-let matrix is $n^k$, with $n$ being the number of letters in the alphabet being used. This means that the size of the k-let matrix can become quite large as $k$ increases. For example, if one were to wish to represent a DNA motif of length 10 as a 10-let, this would require a matrix with 1,048,576 rows (though at this point if what you want is to search for exact sequence matches, the motif format itself is not very useful).

Value

A universal motif object with filled multifreq slot. The bkg slot is also expanded with corresponding higher order probabilities if add.bkg = TRUE.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

scan_sequences(), convert_motifs(), write_motifs()

Examples

sequences <- create_sequences(seqlen = 10)
motif <- create_motif()
motif.trained <- add_multifreq(motif, sequences, add.k = 2:4)
## peek at the 2-let matrix:
motif.trained["multifreq"]$2

ArabidopsisMotif

Arabidopsis motif in universal motif format.

Description

Arabidopsis motif trained from ArabidopsisPromoters using MEME version 4. This motif was generated at the command line using the following command: meme promoters.fa -revcomp -nmotifs 3 -mod anr -dna.

Usage

ArabidopsisMotif

Format

universal motif
Arabidopsis Promoters

Description

50 Arabidopsis promoters, each 1000 bases long. See the "Sequence manipulation and scanning" vignette for an example workflow describing extracting promoter sequences.

Usage

ArabidopsisPromoters

Format

DNAStringSet

compare_motifs

Description

Compare motifs using one of the several available metrics. See the "Motif comparisons and P-values" vignette for detailed information.

Usage

compare_motifs(motifs, compare.to, db.scores, use.freq = 1,
use.type = "PPM", method = "PCC", tryRC = TRUE, min.overlap = 6,
min.mean.ic = 0.25, min.position.ic = 0, relative_entropy = FALSE,
normalise.scores = FALSE, max.p = 0.01, max.e = 10, nthreads = 1,
score.strat = "a.mean", output.report, output.report.max.print = 10)

Arguments

motifs See convert_motifs() for acceptable motif formats.
compare.to numeric If missing, compares all motifs to all other motifs. Otherwise compares all motifs to the specified motif(s).
db.scores data.frame or DataFrame. See details.
use.freq numeric(1). For comparing the multifreq slot.
use.type character(1) One of 'PPM' and 'ICM'. The latter allows for taking into account the background frequencies if relative_entropy = TRUE. Note that 'ICM' is not allowed when method = c("ALLR", "ALLR_LL").
method character(1) One of PCC, EUCIL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
tryRC: logical(1) Try the reverse complement of the motifs as well, report the best score.

min.overlap: numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher than the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.

min.mean.ic: numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use average_ic() to filter out low IC motifs to get around this if you want to avoid getting NAs in your output.

min.position.ic: numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.

relative_entropy: logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic. See convert_type().

normalise.scores: logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.

max.p: numeric(1) Maximum P-value allowed in reporting matches. Only used if compare.to is set.

max.e: numeric(1) Maximum E-value allowed in reporting matches. Only used if compare.to is set. The E-value is the P-value multiplied by the number of input motifs times two.

nthreads: numeric(1) Run compare_motifs() in parallel with nthreads threads. nthreads = 0 uses all available threads.

score.strat: character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

output.report: character(1) Provide a filename for compare_motifs() to write an html output report to. The top matches are shown alongside figures of the match alignments. This requires the knitr and rmarkdown packages. (Note: still in development.)

output.report.max.print: numeric(1) Maximum number of top matches to print.

**Details**

**Available metrics:**

The following metrics are available:
• Euclidean distance (EUCL) (Choi et al. 2004)
• Weighted Euclidean distance (WEUCL)
• Kullback-Leibler divergence (KL) (Kullback and Leibler 1951; Roepcke et al. 2005)
• Hellinger distance (HELL) (Hellinger 1909)
• Squared Euclidean distance (SEUCL)
• Manhattan distance (MAN)
• Pearson correlation coefficient (PCC)
• Weighted Pearson correlation coefficient (WPCC)
• Sandelin-Wasserman similarity (SW), or sum of squared distances (Sandelin and Wasserman 2004)
• Average log-likelihood ratio (ALLR) (Wang and Stormo 2003)
• Lower limit ALLR (ALLR_LL) (Mahony et al. 2007)
• Bhattacharyya coefficient (BHAT) (Bhattacharyya 1943)

Comparisons are calculated between two motifs at a time. All possible alignments are scored, and the best score is reported. In an alignment scores are calculated individually between columns. How those scores are combined to generate the final alignment scores depends on score.strat. See the "Motif comparisons and P-values" vignette for a description of the various metrics. Note that PCC, WPCC, SW, ALLR, ALLR_LL and BHAT are similarities; higher values mean more similar motifs. For the remaining metrics, values closer to zero represent more similar motifs.

Small pseudocounts are automatically added when one of the following methods is used: KL, ALLR, ALLR_LL, IS. This is to avoid zeros in the calculations.

Calculating P-values:
To note regarding p-values: P-values are pre-computed using the make_DBscores() function. If not given, then uses a set of internal precomputed P-values from the JASPAR2018 CORE motifs. These precalculated scores are dependent on the length of the motifs being compared. This takes into account that comparing small motifs with larger motifs leads to higher scores, since the probability of finding a higher scoring alignment is higher.

The default P-values have been precalculated for regular DNA motifs. They are of little use for motifs with a different number of alphabet letters (or even the multifreq slot).

Value
matrix if compare.to is missing; DataFrame otherwise. For the latter, function args are stored in the metadata slot.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

`convert_motifs()`, `motif_tree()`, `view_motifs()`, `make_DBscores()`

Examples

```r
motif1 <- create_motif(name = "1")
motif2 <- create_motif(name = "2")
motif1vs2 <- compare_motifs(c(motif1, motif2), method = "PCC")
## To get a dist object:
as.dist(1 - motif1vs2)

motif3 <- create_motif(name = "3")
motif4 <- create_motif(name = "4")
motifs <- c(motif1, motif2, motif3, motif4)
## Compare motif "2" to all the other motifs:
if (R.Version()$arch != "i386") {
  compare_motifs(motifs, compare.to = 2, max.p = 1, max.e = Inf)
}
## If you are working with a large list of motifs and the mean.min.ic
## option is not set to zero, you may get a number of failed comparisons
## due to low IC. To filter the list of motifs to avoid these, use
## the average_ic() function to remove motifs with low average IC:
## Not run:
```
library(MotifDb)
motifs <- convert_motifs(MotifDb)[1:100]
compare_motifs(motifs)
#> Warning in compare_motifs(motifs) :
#> Some comparisons failed due to low IC

motifs <- motifs[average_ic(motifs) > 0.5]
compare_motifs(motifs)

## End(Not run)

convert_motifs

Convert motif class.

Description

Allows for easy transfer of motif information between different classes as defined by other Bioconductor packages. This function is also used by nearly all other functions in this package, so any motifs of a compatible class can be used without needing to be converted beforehand.

Usage

convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'AsIs'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'list'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'universalmotif'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'MotifList'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'TFFMFirst'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'PFMatrix'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'PWMMatrix'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'ICMatrix'
convert_motifs(motifs, class = "universalmotif-universalmotif")
convert_motifs

## S4 method for signature 'XMatrixList'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'pwm'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'pcm'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'pfm'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'PWM'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'Motif'
convert_motifs(motifs, class = "universalmotif-universalmotif")

## S4 method for signature 'matrix'
convert_motifs(motifs, class = "universalmotif-universalmotif")

Arguments

- **motifs**: Single motif object or list. See details.
- **class**: character(1) Desired motif class. Input as 'package-class'. If left empty, defaults to 'universalmotif-universalmotif'. (See details.)

Details

**Input:**
The following package-class combinations can be used as input:

- MotifDb-MotifList
- TFBSTools-PFMatrix
- TFBSTools-PWMATRIX
- TFBSTools-ICMatrix
- TFBSTools-PFMatrixList
- TFBSTools-PWMATRIXList
- TFBSTools-ICMatrixList
- TFBSTools-TFFMFirst
- seqLogo-pwm
- motifStack-pcm
- motifStack-pfm
- PWMEnrich-PWM
- motifRG-Motif
- universalmotif-universalmotif
Output:
The following package-class combinations can be output:

- MotifDb-MotifList
- TFBSTools-PFMatrix
- TFBSTools-PWMatrix
- TFBSTools-ICMatrix
- TFBSTools-TFFMFirst
- seqLogo-pwm
- motifStack-pcm
- motifStack-pfm
- PWMEnrich-PWM
- Biostrings-PWM (type = 'log2prob')
- rGADEM-motif
- universalmotif-universalmotif (the default, no need to specify this)

Note: MotifDb-MotifList output was a later addition to `convert_motifs()`. As a result, to stay consistent with previous behaviour most functions will always convert MotifDb-MotifList objects to a list of universalmotif motifs, even if other formats would be simply returned as is (e.g. for other formats, `filter_motifs()` will return the input format; for MotifDb-MotifList, a list of universalmotif objects will be returned).

Value

Single motif object or list.

Methods (by class)

- AsIs: Generate an error to remind users to run `to_list()` instead of using the column from `to_df()` directly.
- list: Convert a list of motifs.
- universalmotif: Convert a universalmotif object.
- MotifList: Convert MotifList motifs. (MotifDb)
- TFFMFirst: Convert TFFMFirst motifs. (TFBSTools)
- PFMatrix: Convert PFMatrix motifs. (TFBSTools)
- PWMMatrix: Convert PWMMatrix motifs. (TFBSTools)
- ICMatrix: Convert ICMatrix motifs. (TFBSTools)
- XMatrixList: Convert XMatrixList motifs. (TFBSTools)
- pwm: Convert pwm motifs. (seqLogo)
- pcm: Convert pcm motifs. (motifStack)
- pfm: Convert pfm motifs. (motifStack)
- PWM: Convert PWM motifs. (PWMEnrich)
- Motif: Convert Motif motifs. (motifRG)
- matrix: Create motif from matrices.
convert_motifs

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References

Examples
# Convert from universalmotif:
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
                                package = "universalmotif"))
if (requireNamespace("motifStack", quietly = TRUE)) {
  jaspar.motifstack.pfm <- convert_motifs(jaspar, "motifStack-pfm")
}

# Convert from another class to universalmotif:
if (requireNamespace("TFBSTools", quietly = TRUE)) {
  library(TFBSTools)
data(MA0003.2)
motif <- convert_motifs(MA0003.2)
}

# Convert from another class to another class
if (requireNamespace("PWMEnrich", quietly = TRUE)) {
  motif <- convert_motifs(MA0003.2, "PWMEnrich-PWM")
}

# The 'convert_motifs' function is embedded in the rest of the universalmotif
# functions: non-universalmotif class motifs can be used
MA0003.2.trimmed <- trim_motifs(MA0003.2)
# Note: if the motif object going in has information that the
# 'universalmotif' class can't hold, it will be lost
**Description**

Switch between position count matrix (PCM), position probability matrix (PPM), position weight matrix (PWM), and information count matrix (ICM) types. See the "Introduction to sequence motifs" vignette for details. Please also note that type conversion occurs implicitly throughout the universalmotif package, so there is generally no need to perform this manual conversion. Also please be aware that the message concerning pseudocount-adjusting motifs can be disabled via options(pseudocount.warning=FALSE).

**Usage**

```r
classify_motifs(motifs, type, pseudocount, nsize_correction = FALSE, relative_entropy = FALSE)
```

**Arguments**

- **motifs**: See `classify_motifs()` for acceptable formats.
- **type**: character(1) One of c('PCM', 'PPM', 'PWM', 'ICM').
- **pseudocount**: numeric(1) Correction to be applied to prevent -Inf from appearing in PWM matrices. If missing, the pseudocount stored in the universalmotif 'pseudocount' slot will be used.
- **nsize_correction**: logical(1) If true, the ICM at each position will be corrected to account for small sample sizes. Only used if relative_entropy = FALSE.
- **relative_entropy**: logical(1) If true, the ICM will be calculated as relative entropy. See details.

**Details**

**PCM:**
Position count matrix (PCM), also known as position frequency matrix (PFM). For n sequences from which the motif was built, each position is represented by the numbers of each letter at that position. In theory all positions should have sums equal to n, but not all databases are this consistent. If converting from another type to PCM, column sums will be equal to the 'nsites' slot. If empty, 100 is used.

**PPM:**
Position probability matrix (PPM), also known as position frequency matrix (PFM). At each position, the probability of individual letters is calculated by dividing the count for that letter by the total sum of counts at that position (letter_count / position_total). As a result, each position will sum to 1. Letters with counts of 0 will thus have a probability of 0, which can be undesirable when searching for motifs in a set of sequences. To avoid this a pseudocount can be added ((letter_count + pseudocount) / (position_total + pseudocount)).
PWM:
Position weight matrix (PWM; Stormo et al. (1982)), also known as position-specific weight matrix (PSWM), position-specific scoring matrix (PSSM), or log-odds matrix. At each position, each letter is represented by it's log-likelihood \( \log_2(\text{letter \_probability} / \text{background \_probility}) \), which is normalized using the background letter frequencies. A PWM matrix is constructed from a PPM. If any position has \( 0 \)-probability letters to which pseudocounts were not added, then the final log-likelihood of these letters will be \(-\infty\).

ICM:
Information content matrix (ICM; Schneider and Stephens 1990). An ICM is a PPM where each letter probability is multiplied by the total information content at that position. The information content of each position is determined as: \( \text{totalIC} \leftarrow \text{Hi} \), where the total information \( \text{totalIC} \) \( \text{totalIC} \leftarrow \log_2(\text{alphabet \_length}) \), and the Shannon entropy (Shannon 1948) for a specific position (Hi)
\[ \text{Hi} \leftarrow -\sum(\text{apply(\text{alphabet \_frequencies}, \text{function(x) x * log(2)})}). \]

As a result, the total sum or height of each position is representative of it's sequence conservation, measured in the unit 'bits', which is a unit of energy (Schneider 1991; see https://fr-s-schneider.ncifcrf.gov/logorecommendations.html for more information). However not all programs will calculate information content the same. Some will 'correct' the total information content at each position using a correction factor as described by Schneider et al. (1986). This correction can applied by setting \text{nsize\_correction = TRUE}, however it will only be applied if the 'nsites' slot is not empty. As such, converting from an ICM to which some form of correction has been applied will result in a PCM/PPM/PWM with slight inaccuracies.

Another method of calculating information content is calculating the relative entropy, also known as Kullback-Leibler divergence (Kullback and Leibler 1951). This accounts for background frequencies, which can be useful for genomes with a heavy imbalance in letter frequencies. For each position, the individual letter frequencies are calculated as \( \text{letter \_freq} \times \log_2(\text{letter \_freq} / \text{bkg \_freq}) \). When calculating information content using Shannon entropy, the maximum content for each position will always be \( \log_2(\text{alphabet \_length}) \). This does not hold for information content calculated as relative entropy. Please note that conversion from ICM assumes the information content was not calculated as relative entropy.

Value
See \text{convert\_motifs()} for possible output motif objects.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

convert_motifs()

Examples

```r
jaspar.pcm <- read_jaspar(system.file("extdata", "jaspar.txt", 
package = "universalmotif"))

## The motifs pseudocounts are 1: these will be used in the PCM->PPM 
## calculation
jaspar.pwm <- convert_type(jaspar.pcm, type = "PPM")

## Setting pseudocount to 0 will prevent any correction from being 
## applied to PPM/PWM matrices, overriding the motifs own pseudocounts
jaspar.pwm <- convert_type(jaspar.pcm, type = "PWM", pseudocount = 0)
```

create_motif

Create a motif.

Description

Create a motif from a set of sequences, a matrix, or generate a random motif. See the "Motif import, export and manipulation" vignette for details.

Usage

```r
create_motif(input, alphabet, type = "PPM", name = "motif", 
pseudocount = 0, bkg, nsites, altname, family, organism, bkgsites, strand, 
pval, qval, eval, extrainfo, add.multifreq)
```

## S4 method for signature 'missing'
create_motif(input, alphabet, type = "PPM",
create_motif

name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'numeric'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'character'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'matrix'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'DNAStringSet'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'RNAStringSet'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'AAStringSet'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

## S4 method for signature 'BStringSet'
create_motif(input, alphabet, type = "PPM",
             name = "motif", pseudocount = 0, bkg, nsites, alname, family, organism, bkgsites, strand, pval, qval, eval, extrainfo, add.multifreq)

Arguments

input character, numeric, matrix, XStringSet, or missing.
alphabet character(1) One of c('DNA', 'RNA', 'AA'), or a combined string representing the letters. If no alphabet is provided then it will try and guess the alphabet from the input.
type character(1) One of c('PCM', 'PPM', 'PWM', 'ICM').
name character(1) Motif name.
pseudocount numeric(1) Correction to be applied to prevent -Inf from appearing in PWM matrices. Defaults to 0.
### create_motif

- **bkg** numeric A vector of probabilities, each between 0 and 1. If higher order backgrounds are provided, then the elements of the vector must be named. If unnamed, then the order of probabilities must be in the same order as the alphabetically sorted sequence alphabet.

- **nsites** numeric(1) Number of sites the motif was constructed from. If blank, then `create_motif()` will guess the appropriate number if possible. To prevent this, provide `nsites = numeric()`.

- **altname** character(1) Alternate motif name.

- **family** character(1) Transcription factor family.

- **organism** character(1) Species of origin.

- **bkgsites** numeric(1) Total number of sites used to find the motif.

- **strand** character(1) Whether the motif is specific to a certain strand. Acceptable strands are '+' , '-' , and '+' (to represent both strands). Note that '-' and '*' can also be provided to represent both strands, but the final strand in the `universalmotif` object will be set to '+' .

- **pval** numeric(1) P-value associated with motif.

- **qval** numeric(1) Adjusted P-value associated with motif.

- **eval** numeric(1) E-value associated with motif.

- **extrainfo** character Any other extra information, represented as a named character vector.

- **add.multifreq** numeric If the motif is created from a set of sequences, then the `add_multifreq()` function can be run at the same time (with `RC = FALSE`).

### Details

The aim of this function is provide an easy interface to creating `universalmotif` motifs, as an alternative to the default class constructor (i.e. `new('universalmotif', name=...)`). See examples for potential use cases.

Note: when generating random motifs, the `nsites` slot is also given a random value.

See the examples section for more info on motif creation.

### Value

- `universalmotif` object.

### Methods (by class)

- **missing**: Create a random motif of length 10.
- **numeric**: Create a random motif with a specified length.
- **character**: Create motif from a consensus string.
- **matrix**: Create motif from a matrix.
- **DNAStringSet**: Create motif from a `DNAStringSet`.
- **RNAStringSet**: Create motif from a `RNAStringSet`.
- **AAStringSet**: Create motif from a `AAStringSet`.
- **BStringSet**: Create motif from a `BStringSet`. 

---

**create_motif**
Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also
convert_type(), add_multifreq(), create_sequences(), shuffle_motifs().

Examples

##### create motifs from a single string

# Motif is by default generated as a PPM: change final type as desired
DNA.motif <- create_motif("TATAWAW")
DNA.motif <- create_motif("TATAWAW", type = "PCM")

# Nsites will be set to the number of input sequences unless specified or
# a single string is used as input
DNA.motif <- create_motif("TTTTTTT", nsites = 10)

# Ambiguity letters can be used:
DNA.motif <- create_motif("TATAWAW")
DNA.motif <- create_motif("NNVVWWAAWWDDN")

# Be careful about setting nsites when using ambiguity letters!
DNA.motif <- create_motif("NNVVWWWAAWWDDN", nsites = 1)

RNA.motif <- create_motif("UUUCCG")

# 'create_motif' will try to detect the alphabet type; this can be
# unreliable for AA and custom alphabets as DNA and RNA alphabets are
# detected first
AA.motif <- create_motif("AVLK", alphabet = "AA")
custom.motif <- create_motif("QWER", alphabet = "QWER")
# Specify custom alphabet
custom.motif <- create_motif("QWER", alphabet = "QWERASDF")

###### Create motifs from multiple strings of equal length

DNA.motif <- create_motif(c("TTTT", "AAAA", "AACC", "TTGG"), type = "PPM")
DNA.motif <- create_motif(c("TTTT", "AAAA", "AACC", "TTGG"), nsites = 20)
RNA.motif <- create_motif(c("UUUU", "AAAA", "AACC", "UUGG"), type = "PWM")
AA.motif <- create_motif(c("ARNDCQ", "EGHILK", "ARNDCQ"), alphabet = "AA")
custom.motif <- create_motif(c("POIU", "LKJH", "POIU", "CVBN"),
                           alphabet = "POIULKJHCVBN")

# Ambiguity letters are only allowed for single consensus strings: the
# following fails
## Not run:
create_motif(c("WWTT", "CCGG"))
create_motif(c("XXXX", "XXXX"), alphabet = "AA")
library(Biostrings)

DNA.set <- DNAStringSet(c("TTTT", "AAAA", "AACC", "TTGG"))
DNA.motif <- create_motif(DNA.set)
RNA.set <- RNAStringSet(c("UUUU", "AACC", "UUCC"))
RNA.motif <- create_motif(RNA.set)
AA.set <- AAStringSet(c("VVVLLL", "AAAIII"))
AA.motif <- create_motif(AA.set)

# Custom motifs can be created from BStringSet objects
B.set <- BStringSet(c("QWER", "ASDF", "ZXCV", "TYUI"))
custom.motif <- create_motif(B.set)

DNA.motif.k2 <- create_motif(DNA.set, add.multifreq = 2)

##### Create motifs from matrices

mat <- matrix(c(1, 1, 1, 1,
                 2, 0, 2, 0,
                 0, 2, 0, 2,
                 0, 0, 0, 0),
               nrow = 4, byrow = TRUE)
DNA.motif <- create_motif(mat, alphabet = "DNA")
RNA.motif <- create_motif(mat, alphabet = "RNA", nsites = 20)
custom.motif <- create_motif(mat, alphabet = "QWER")

# Specify custom alphabet
custom.motif <- create_motif(mat, alphabet = "QWER")

# Alphabet can be detected from rownames
rownames(mat) <- DNA_BASES
DNA.motif <- create_motif(mat)
rownames(mat) <- c("Q", "W", "E", "R")
custom.motif <- create_motif(mat)

# Matrices can also be used as input
mat.ppm <- matrix(c(0.1, 0.1, 0.1, 0.1,
                     0.5, 0.5, 0.5, 0.5,
                     0.1, 0.1, 0.1, 0.1,
                     0.3, 0.3, 0.3, 0.3),
                       nrow = 4, byrow = TRUE)
DNA.motif <- create_motif(mat.ppm, alphabet = "DNA", type = "PPM")

##### Create random motifs

# These are generated as PPMs with 10 positions
create_sequences <- create_motif()
RNA.motif <- create_motif(alphabet = "RNA")
AA.motif <- create_motif(alphabet = "AA")
custom.motif <- create_motif(alphabet = "QWER")

# The number of positions can be specified
DNA.motif <- create_motif(5)

# If the background frequencies are not provided, they are generated
# using `rpois`; positions are created using `rdirichlet(1, bkg)`.
# (calling `create_motif()` creates motifs with an average
# positional IC of 1)
DNA.motif <- create_motif(bkg = c(0.3, 0.2, 0.2, 0.3))
DNA.motif <- create_motif(10, bkg = c(0.1, 0.4, 0.4, 0.1))

---

create_sequences  
Create random sequences.

Description

Generate random sequences from any set of characters, represented as XStringSet objects.

Usage

create_sequences(alphabet = "DNA", seqnum = 100, seqlen = 100, freqs, nthreads = 1, rng.seed = sample.int(10000, 1))

Arguments

- **alphabet** `character(1)` One of c("DNA", "RNA", "AA"), or a string of characters to be used as the alphabet.
- **seqnum** `numeric(1)` Number of sequences to generate.
- **seqlen** `numeric(1)` Length of random sequences.
- **freqs** `numeric` A named vector of probabilities. The length of the vector must be the power of the number of letters in the sequence alphabet. Probabilities can only be provided for a single size k.
- **nthreads** `numeric(1)` Run `create_sequences()` in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with seqnum = 1.
- **rng.seed** `numeric(1)` Set random number generator seed. Since sequence creation can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Each individual sequence creation instance is given the following
seed: \texttt{rng.seed * index}. The default is to pick a random number as chosen by \texttt{sample()}, which effectively is making \texttt{create_sequences()} dependent on the R RNG state.

\textbf{Value}

\texttt{XStringSet} The returned sequences are \textit{unnamed}.

\textbf{Author(s)}

Benjamin Jean-Marie Tremblay, \texttt{<benjamin.tremblay@uwaterloo.ca>}

\textbf{See Also}

\texttt{create_motif()}, \texttt{shuffle_sequences()}

\textbf{Examples}

```r
## Create DNA sequences with slightly increased AT content:
sequences <- create_sequences(freqs = c(A=0.3, C=0.2, G=0.2, T=0.3))
## Create custom sequences:
sequences.QWER <- create_sequences("QWER")
## You can include non-alphabet characters as well, even spaces:
sequences.custom <- create_sequences("!@#$ ")
```
Arguments

motifs See convert_motifs() for acceptable motif formats.
sequences XStringSet Sequences to scan. Alphabet should match motif.
bkg.sequences XStringSet Optional. If missing, shuffle_sequences() is used to create background sequences from the input sequences.
max.p numeric(1) P-value threshold.
max.q numeric(1) Adjusted P-value threshold. This is only useful if multiple motifs are being enriched for.
max.e numeric(1). The E-value is calculated by multiplying the P-value with the number of input motifs times two (McLeay and Bailey 2010).
qval.method character(1) See stats::p.adjust().
threshold numeric(1) See details.
threshold.type character(1) One of c('pvalue', 'qvalue', 'logodds', 'logodds.abs'). See details.
verbose numeric(1) 0 for no output, 4 for max verbosity.
RC logical(1) If TRUE, check reverse complement of the input sequences. Only available for DNA/RNA.
use.freq numeric(1) The default, 1, uses the motif matrix (from the motif['motif'] slot) to search for sequences. If a higher number is used, then the matching k-let matrix from the motif['multifreq'] slot is used. See add_multifreq().
shuffle.k numeric(1) The k-let size to use when shuffling input sequences. Only used if no background sequences are input. See shuffle_sequences().
shuffle.method character(1) One of c('euler', 'markov', 'linear'). See shuffle_sequences().
return.scan.results logical(1) Return output from scan_sequences(). For large jobs, leaving this as FALSE can save a small amount time by preventing construction of the complete results data.frame from scan_sequences().
nthreads numeric(1) Run scan_sequences() in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with only a single motif and sequence.
rng.seed numeric(1) Set random number generator seed. Since shuffling can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Each individual sequence in an XStringSet object will be given the following seed: rng.seed * index. See shuffle_sequences().
motif_pvalue.k numeric(1) Control motif_pvalue() approximation. See motif_pvalue().
use.gaps logical(1) Set this to FALSE to ignore motif gaps, if present.
allow.nonfinite logical(1) If FALSE, then apply a pseudocount if non-finite values are found in the PWM. Note that if the motif has a pseudocount greater than zero and the motif is not currently of type PWM, then this parameter has no effect as the pseudocount will be applied automatically when the motif is converted to a
PWM internally. This value is set to FALSE by default in order to stay consistent with pre-version 1.8.0 behaviour. A message will be printed if a pseudocount is applied. To disable this, set options(pseudocount.warning=FALSE).

warn.NA

logical(1) Whether to warn about the presence of non-standard letters in the input sequence, such as those in masked sequences.

no.overlaps

logical(1) Remove overlapping hits from the same motifs. Overlapping hits from different motifs are preserved. Please note that the current implementation of this feature can add significantly to the run time for large inputs.

no.overlaps.by.strand

logical(1) Whether to discard overlapping hits from the opposite strand (TRUE), or to only discard overlapping hits on the same strand (FALSE).

no.overlaps.strat

categorical(1) One of c("score", "order"). The former option keeps the highest scoring overlapping hit (and the first of these within ties), and the latter simply keeps the first overlapping hit.

respect.strand

logical(1) If motifs are DNA/RNA, then setting this option to TRUE will make scan_sequences() only scan the strands of the input sequences as indicated in the motif strand slot.

motif_pvalue.method

categorical(1) One of c("dynamic", "exhaustive"). Algorithm used for calculating P-values. The "exhaustive" method involves finding all possible motif matches at or above the specified score using a branch-and-bound algorithm, which can be computationally intensive (Hartman et al., 2013). Additionally, the computation must be repeated for each hit. The "dynamic" method calculates the distribution of possible motif scores using a much faster dynamic programming algorithm, and can be recycled for multiple scores (Grant et al., 2011). The only disadvantage is the inability to use allow.nonfinite = TRUE. See motif_pvalue() for details.

scan_sequences.qvals.method

categorical(1) One of c("fdr", "BH", "bonferroni"). The method for calculating adjusted P-values for individual motif hits. These are described in depth in the Sequence Searches vignette.

mode

categorical(1) One of c("total.hits", "seq.hits"). The former enriches for the total count of motif hits across all sequences, whereas the latter only counts motif hits once per sequence (useful for cases where there are many small sequences).

pseudocount

integer(1) Add a pseudocount to the motif hit counts when performing the Fisher test.

Details

To find enriched motifs, scan_sequences() is run on both target and background sequences. stats::fisher.test() is run to test for enrichment.

See scan_sequences() for more info on scanning parameters.
Value

DataFrame Enrichment results in a DataFrame. Function args and (optionally) scan results are stored in the metadata slot.

Author(s)

Benjamin Jean-Marie Tremblay <benjamin.tremblay@uwaterloo.ca>

References


See Also

`scan_sequences()`, `shuffle_sequences()`, `add_multifreq()`, `motif_pvalue()`

Examples

```r
data(ArabidopsisPromoters)
data(ArabidopsisMotif)
if (R.Version()$arch != "i386") {
enrich_motifs(ArabidopsisMotif, ArabidopsisPromoters, threshold = 0.01)
}
```

---

**examplemotif**  
*Example motif in universalmotif format.*

Description

A simple DNA motif. To recreate this motif: `create_motif("TATAWAW", nsites = numeric())`

Usage

`examplemotif`

Format

`universalmotif`
Description

A simple DNA motif with a non-empty multifreq slot. To recreate this motif: add_multifreq(examplemotif,
DNASTRINGSET(rep(c("CAAAACC", "CTTTTCC"), 3)))

Usage

examplemotif2

Format

universalmotif

filter_motifs

Filter a list of motifs.

Description

Filter motifs based on the contents of available universalmotif slots. If the input motifs are not of
universalmotif, then they will be converted for the duration of the filter_motifs() operation.

Usage

filter_motifs(motifs, name, altnames, family, organism, width, alphabet, type,
icscore, nsites, strand, pval, qval, eval, extrainfo)

Arguments

motifs list See convert_motifs() for acceptable formats.
name character Keep motifs by names.
altnames character Keep motifs by altnames.
family character Keep motifs by family.
organism character Keep motifs by organism.
width numeric(1) Keep motifs with minimum width.
alphabet character Keep motifs by alphabet.
type character Keep motifs by type.
icscore numeric(1) Keep motifs with minimum total IC.
nsites numeric(1) Keep motifs with minimum number of target sites.
strand character Keeps motifs by strand.
pval numeric(1) Keep motifs by max P-value.
qval numeric(1) Keep motifs by max Q-value.
eval numeric(1) Keep motifs by max E-val.
extrainfo character Named character vector of items that must be present in motif extrainfo slots.

Value

list Motifs. An attempt will be made to preserve the original class, see convert_motifs() for limitations.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

Examples

## By minimum IC:
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
package = "universalmotif"))
jaspar.ic3 <- filter_motifs(jaspar, icscore = 3)

## Starting from version 1.10.0 of the universalmotif package, one
## could instead make use of the universalmotif_df structure:
jaspar.ic3 <- jaspar |> to_df() |> subset(icscore > 3) |> to_list()

## By organism:
## Not run:
library(MotifDb)
motifs <- convert_motifs(MotifDb)
motifs <- filter_motifs(motifs, organism = c("Athaliana", "Mmuscuslus"),
extrainfo = c("dataSource" = "cisbp_1.02"))

## Or:
motifs <- convert_motifs(MotifDb) |> to_df() |
subset(organism %in% c("Athaliana", "Mmuscuslus") &
dataSource == "cisbp_1.02") |> to_list()

## End(Not run)

fontDFroboto Polygon coordinates for plotting letters.

Description

DataFrame of polygon coordinates used by view_motifs() for plotting letters. It was generated using the createPolygons function from the gglogo package for the font Roboto Medium.
Usage

fontDFroboto

Format

DataFrame

get_bkg

Calculate sequence background.

Description

For a set of input sequences, calculate the overall sequence background for any k-let size. For very large sequences DNA and RNA sequences (in the billions of bases), please be aware of the much faster and more efficient Biostrings::oligonucleotideFrequency(). get_bkg() can still be used in these cases, though it may take several seconds or minutes to calculate the results (depending on requested k-let sizes).

Usage

get_bkg(sequences, k = 1:3, as.prob = NULL, pseudocount = 0, alphabet = NULL, to.meme = NULL, RC = FALSE, list.out = NULL, nthreads = 1, merge.res = TRUE, window = FALSE, window.size = 0.1, window.overlap = 0)

Arguments

sequences XStringSet Input sequences. Note that if multiple sequences are present, the results will be combined into one (unless merge.res = FALSE).

k integer Size of k-let. Background can be calculated for any k-let size.

as.prob integer(1) Add a count to each possible k-let. Prevents any k-let from having 0 or 1 probabilities.

pseudocount integer(1) Add a count to each possible k-let. Prevents any k-let from having 0 or 1 probabilities.

alphabet character(1) Provide a custom alphabet to calculate a background for. If NULL, then standard letters will be assumed for DNA, RNA and AA sequences, and all unique letters found will be used for BStringSet type sequences. Note that letters which are not a part of the standard DNA/RNA/AA alphabets or in the provided alphabet will not be counted in the totals during probability calculations.

to.meme character(1) Provide a custom alphabet to calculate a background for. If NULL, then standard letters will be assumed for DNA, RNA and AA sequences, and all unique letters found will be used for BStringSet type sequences. Note that letters which are not a part of the standard DNA/RNA/AA alphabets or in the provided alphabet will not be counted in the totals during probability calculations.

RC logical(1) Calculate the background of the reverse complement of the input sequences as well. Only valid for DNA/RNA.
get_bkg

- `list.out`: Deprecated.
- `nthreads`: `numeric(1)` Run `get_bkg()` in parallel with `nthreads` threads. `nthreads = 0` uses all available threads. Note that no speed up will occur for jobs with only a single sequence.
- `merge.res`: `logical(1)` Whether to merge results from all sequences or return background data for individual sequences.
- `window`: `logical(1)` Determine background in windows.
- `window.size`: `numeric` Window size. If a number between 0 and 1 is provided, the value is calculated as the number multiplied by the sequence length.
- `window.overlap`: `numeric` Overlap between windows. If a number between 0 and 1 is provided, the value is calculated as the number multiplied by the sequence length.

**Value**

If `to.meme = NULL`, a `DataFrame` with columns `klet`, `count`, and `probability`. If `merge.res = FALSE`, there will be an additional sequence column. If `window = TRUE`, there will be an additional `start` and `stop` columns.

If `to.meme` is not `NULL`, then `NULL` is returned, invisibly.

**Author(s)**

Benjamin Jean-Marie Tremblay, `<benjamin.tremblay@uwaterloo.ca>`

**References**


**See Also**

`create_sequences()`, `scan_sequences()`, `shuffle_sequences()`

**Examples**

```r
## Compare to Biostrings version
library(Biostrings)
seqs.DNA <- create_sequences()
bkg.DNA <- get_bkg(seqs.DNA, k = 3)
bkg.DNA2 <- oligonucleotideFrequency(seqs.DNA, 3, 1, as.prob = FALSE)
bkg.DNA2 <- colSums(bkg.DNA2)
all(bkg.DNA$count == bkg.DNA2)

## Create a MEME background file
get_bkg(seqs.DNA, k = 1:3, to.meme = stdout(), pseudocount = 1)

## Non-DNA/RNA/AA alphabets
seqs.QWERTY <- create_sequences("QWERTY")
bkg.QWERTY <- get_bkg(seqs.QWERTY, k = 1:2)
```
**Description**

For use with `compare_motifs()`. The precomputed scores allow for fast P-value estimation. These scores were generated using `make_DBscores()` with the JASPAR2018 CORE motif set. The scores are organized in a DataFrame. In this DataFrame is the location and scale of scores resulting from a statistical distribution using the comparisons of JASPAR2018 CORE motifs with randomized motifs of the specified subject and target motif length. Created using `make_DBscores()` from universalmotif v1.4.0. The parameters used can be seen via `S4Vectors::metadata(JASPAR2018_CORE_DBSCORES)`.

**Usage**

`JASPAR2018_CORE_DBSCORES`

**Format**

DataFrame with function args in the metadata slot.

---

**make_DBscores**  
*Create P-value databases.*

**Description**

Generate data used by `compare_motifs()` for P-value calculations. By default, `compare_motifs()` uses an internal database based on the JASPAR2018 core motifs (Khan et al. 2018). Parameters for distributions are estimated for every combination of motif widths.

**Usage**

```r
make_DBscores(db.motifs, method = c("PCC", "EUCL", "SW", "KL", "WEUCL", "ALLR", "BHAT", "HELL", "WPCC", "SEUCL", "MAN", "ALLR_LL"), shuffle.db = TRUE, shuffle.k = 3, shuffle.method = "linear", rand.tries = 1000, widths = 5:30, min.position.ic = 0, normalise.scores = c(FALSE, TRUE), min.overlap = 6, min.mean.ic = 0.25, progress = TRUE, nthreads = 1, tryRC = TRUE, score.strat = c("sum", "a.mean", "g.mean", "median", "wa.mean", "wg.mean", "fzt"))
```
**make_DBscores**

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db.motifs</td>
<td>list Database motifs.</td>
</tr>
<tr>
<td>method</td>
<td>character(1) One of PCC, EUCL, SW, KL, ALLR, BHA, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.</td>
</tr>
<tr>
<td>shuffle.db</td>
<td>logical(1) Deprecated. Does nothing. generate random motifs with create_motif().</td>
</tr>
<tr>
<td>shuffle.k</td>
<td>numeric(1) See shuffle_motifs().</td>
</tr>
<tr>
<td>shuffle.method</td>
<td>character(1) See shuffle_motifs().</td>
</tr>
<tr>
<td>rand.tries</td>
<td>numeric(1) Approximate number of comparisons to perform for every combination of widths.</td>
</tr>
<tr>
<td>widths</td>
<td>numeric Motif widths to use in P-value database calculation.</td>
</tr>
<tr>
<td>min.position.ic</td>
<td>numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.</td>
</tr>
<tr>
<td>normalise.scores</td>
<td>logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.</td>
</tr>
<tr>
<td>min.overlap</td>
<td>numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.</td>
</tr>
<tr>
<td>min.mean.ic</td>
<td>numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use average_ic() to filter out low IC motifs to get around this if you want to avoid getting NAs in your output.</td>
</tr>
<tr>
<td>progress</td>
<td>logical(1) Show progress.</td>
</tr>
<tr>
<td>nthreads</td>
<td>numeric(1) Run compare_motifs() in parallel with nthreads threads. nthreads = 0 uses all available threads.</td>
</tr>
<tr>
<td>tryRC</td>
<td>logical(1) Try the reverse complement of the motifs as well, report the best score.</td>
</tr>
<tr>
<td>score.strat</td>
<td>character(1) How to handle column scores calculated from motif alignments. &quot;sum&quot;: add up all scores. &quot;a.mean&quot;: take the arithmetic mean. &quot;g.mean&quot;: take the geometric mean. &quot;median&quot;: take the median. &quot;wa.mean&quot;, &quot;wg.mean&quot;: weighted arithmetic/geometric mean. &quot;fzt&quot;: Fisher Z-transform. Weights are the total information content shared between aligned columns.</td>
</tr>
</tbody>
</table>

### Details

See compare_motifs() for more info on comparison parameters.
To replicate the internal `universalmotif` DB scores, run `make_DBscores()` with the default settings. Note that this will be a slow process.

Arguments `widths`, `method`, `normalise.scores` and `score.strat` are vectorized; all combinations will be attempted.

Value

A DataFrame with score distributions for the input database. If more than one `make_DBscores()` run occurs (i.e. `args` `method`, `normalise.scores` or `score.strat` are longer than 1), then the function `args` are included in the metadata slot.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

`compare_motifs()`

Examples

```r
## Not run:
library(MotifDb)
motifs <- convert_motifs(MotifDb[1:100])
scores <- make_DBscores(motifs, method = "PCC")
compare_motifs(motifs, 1:100, db.scores = scores)
## End(Not run)
```

Description

Aligns the motifs using `compare_motifs()`, then averages the motif PPMs. Currently the `multifreq` slot, if filled in any of the motifs, will be dropped. Only 0-order background probabilities will be kept. Motifs are merged one at a time, starting with the first entry in the list.
merge_motifs

Usage

merge_motifs(motifs, method = "ALLR", use.type = "PPM", min.overlap = 6,
min.mean.ic = 0.25, tryRC = TRUE, relative_entropy = FALSE,
normalise.scores = FALSE, min.position.ic = 0, score.strat = "sum",
new.name = NULL)

Arguments

motifs See convert_motifs() for acceptable motif formats.
method character(1) One of PCC, EUCL, SW, KL, ALLR, BHAL, HELL, SEUCL,
MAN, ALLR_LL, WEUCL, WPCC. See details.
use.type character(1) One of 'PPM' and 'ICM'. The latter allows for taking into account
the background frequencies if relative_entropy = TRUE. Note that 'ICM'
is not allowed when method = c("ALLR", "ALLR_LL").
min.overlap numeric(1) Minimum overlap required when aligning the motifs. Setting this
to a number higher then the width of the motifs will not allow any overhangs.
Can also be a number between 0 and 1, representing the minimum fraction that the
motifs must overlap.
min.mean.ic numeric(1) Minimum mean information content between the two motifs for
an alignment to be scored. This helps prevent scoring alignments between
low information content regions of two motifs. Note that this can result in some
comparisons failing if no alignment passes the mean IC threshold. Use
average_ic() to filter out low IC motifs to get around this if you want to avoid
getting NAs in your output.
tryRC logical(1) Try the reverse complement of the motifs as well, report the best
score.
relative_entropy logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic.
See convert_type().
normalise.scores logical(1) Favour alignments which leave fewer unaligned positions, as well
as alignments between motifs of similar length. Similarity scores are multiplied
by the ratio of aligned positions to the total number of positions in the larger
motif, and the inverse for distance scores.
min.position.ic numeric(1) Minimum information content required between individual align-
ment positions for it to be counted in the final alignment score. It is recom-
ended to use this together with normalise.scores = TRUE, as this will help
punish scores resulting from only a fraction of an alignment.
score.strat character(1) How to handle column scores calculated from motif alignments.
"sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean":
take the geometric mean. "median": take the median. "wa.mean", "wg.mean":
weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the
total information content shared between aligned columns.
new.name character(1), NULL Instead of collapsing existing names (if NULL), assign a
new one manually for the merged motif.
merge_similar

Details

See `compare_motifs()` for more info on comparison parameters.

If using a comparison metric where 0s are not allowed (KL, ALLR, ALLR_LL, IS), then pseudocounts will be added internally. These pseudocounts are only used for comparison and alignment, and are not used in the final merging step.

Note: score.strat = "a.mean" is NOT recommended, as `merge_motifs()` will not discriminate between two alignments with equal mean scores, even if one alignment is longer than the other.

Value

A single motif object. See `convert_motifs()` for available formats.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

`compare_motifs()`

Examples

```r
## Not run:
library(MotifDb)
merged.motif <- merge_motifs(MotifDb[1:5])

## End(Not run)

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

merge_similar  Identify and merge similar motifs within a collection of motifs (or simply cluster motifs).

Description

Given a list of motifs, `merge_similar()` will identify similar motifs with `compare_motifs()`, and merge similar ones with `merge_motifs()`.
merge_similar

**Usage**

```r
merge_similar(motifs, threshold = 0.95, threshold.type = "score.abs",
method = "PCC", use.type = "PPM", min.overlap = 6, min.mean.ic = 0,
tryRC = TRUE, relative_entropy = FALSE, normalise.scores = FALSE,
min.position.ic = 0, score.strat.compare = "a.mean",
score.strat.merge = "sum", nthreads = 1, return.clusters = FALSE)
```

**Arguments**

- **motifs**
  See `convert_motifs()` for acceptable motif formats.
- **threshold**
  numeric(1) The minimum (for similarity metrics) or maximum (for distance metrics) threshold score for merging.
- **threshold.type**
  character(1) Type of score used for thresholding. Currently unused.
- **method**
  character(1) One of PCC, EUCL, SW, KL, BHAT, HELL, SEUCL, MAN, WEUCL, WPCC. See `compare_motifs()`. (The ALLR and ALLR_LL methods cannot be used for distance matrix construction.)
- **use.type**
  character(1) One of 'PPM' and 'ICM'. The latter allows for taking into account the background frequencies if `relative_entropy = TRUE`. Note that 'ICM' is not allowed when `method = c("ALLR", "ALLR_LL")`.
- **min.overlap**
  numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher than the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
- **min.mean.ic**
  numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use `average_ic()` to filter out low IC motifs to get around this if you want to avoid getting NAs in your output.
- **tryRC**
  logical(1) Try the reverse complement of the motifs as well, report the best score.
- **relative_entropy**
  logical(1) Change the ICM calculation affecting `min.position.ic` and `min.mean.ic`. See `convert_type()`.
- **normalise.scores**
  logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
- **min.position.ic**
  numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with `normalise.scores = TRUE`, as this will help punish scores resulting from only a fraction of an alignment.
motif_peaks

Look for overrepresented motif position peaks in a set of sequences.

Description

Using the motif position data from scan_sequences() (or elsewhere), test whether certain positions in the sequences have significantly higher motif density.

score.strat.compare

character(1) The score.strat parameter used by compare_motifs(). For clustering purposes, the "sum" option cannot be used.

score.strat.merge

character(1) The score.strat parameter used by merge_motifs(). As discussed in merge_motifs(), the "sum" option is recommended over "a.mean" to maximize the overlap between motifs.

nthreads

numeric(1) Run compare_motifs() in parallel with nthreads threads. nthreads = 0 uses all available threads.

return.clusters

logical(1) Return the clusters instead of merging.

Details

See compare_motifs() for more info on comparison parameters, and merge_motifs() for more info on motif merging.

Value

See convert_motifs() for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

compare_motifs(), merge_motifs()

Examples

## Not run:
library(MotifDb)
motifs <- filter_motifs(MotifDb, family = "bHLH")[1:50]
length(motifs)
motifs <- merge_similar(motifs)
length(motifs)

## End(Not run)
motif_peaks

Usage

motif_peaks(hits, seq.length, seq.count, bandwidth, max.p = 1e-06,
peak.width = 3, nrand = 100, plot = TRUE, BP = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hits</td>
<td>numeric A vector of sequence positions indicating motif sites.</td>
</tr>
<tr>
<td>seq.length</td>
<td>numeric(1) Length of sequences. Only one number is allowed, as all sequences must be of identical length. If missing, then the largest number from hits is used.</td>
</tr>
<tr>
<td>seq.count</td>
<td>numeric(1) Number of sequences with motif sites. If missing, then the number of unique values in hits is used.</td>
</tr>
<tr>
<td>bandwidth</td>
<td>numeric(1) Peak smoothing parameter. Smaller numbers will result in skinnier peaks, larger numbers will result in wider peaks. Leaving this empty will cause motif_peaks() to generate one by itself (see ‘details’).</td>
</tr>
<tr>
<td>max.p</td>
<td>numeric(1) Maximum P-value allowed for finding significant motif site peaks.</td>
</tr>
<tr>
<td>peak.width</td>
<td>numeric(1) Minimum peak width. A peak is defined as the highest point within the value set by peak.width.</td>
</tr>
<tr>
<td>nrand</td>
<td>numeric(1) Number of random permutations for generating a null distribution. In order to calculate P-values, a set of random motif site positions are generated nrand times.</td>
</tr>
<tr>
<td>plot</td>
<td>logical(1) Will create a ggplot2 object displaying motif peaks.</td>
</tr>
<tr>
<td>BP</td>
<td>logical(1) Allows for the use of BiocParallel within motif_peaks(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for exceptionally large jobs. Keep in mind that this function will not attempt to limit its memory usage.</td>
</tr>
</tbody>
</table>

Details

Kernel smoothing is used to calculate motif position density. The implementation for this process is based on code from the KernSmooth R package (Wand 2015). These density estimates are used to determine peak locations and heights. To calculate the P-values of these peaks, a null distribution is calculated from peak heights of randomly generated motif positions.

If the bandwidth option is not supplied, then the following code is used (from KernSmooth):

del0 <- (1 / (4 * pi))^(1 / 10)
bandwidth <- del0 * (243 / (35 * length(hits)))^(1 / 5) * sqrt(var(hits))

Value

A DataFrame with peak positions and P-values. If plot = TRUE, then a list is returned with the DataFrame as the first item and the ggplot2 object as the second item.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>
motif_pvalue

Motif P-value and scoring utility

Description

For calculating P-values and logodds scores from P-values for any number of motifs.

Usage

```r
motif_pvalue(motifs, score, pvalue, bkg.probs, use.freq = 1, k = 8,
             nthreads = 1, rand.tries = 10, rng.seed = sample.int(10000, 1),
             allow.nonfinite = FALSE, method = c("dynamic", "exhaustive"))
```

Arguments

- **motifs**: See `convert_motifs()` for acceptable motif formats.
- **score**: numeric, list Get a P-value for a motif from a logodds score. See details for an explanation of how to vectorize the calculation for method = "dynamic".
- **pvalue**: numeric, list Get a logodds score for a motif from a P-value. See details for an explanation of how to vectorize the calculation for method = "dynamic".
- **bkg.probs**: numeric, list A vector background probabilities. If supplying individual background probabilities for each motif, a list of such vectors. If missing, retrieves the background from the motif bkg slot. Note that this option is only used when method = "dynamic", or when method = "exhaustive" and providing a
P-value and returning a score; for the inverse, the motifs are first converted to PWMs via `convert_type()`, which uses the motif bkg slot for background adjustment.

**P-value and returning a score:**

- **use.freq** numeric(1) By default uses the regular motif matrix; otherwise uses the corresponding multifreq matrix. Max is 3 when method = "exhaustive".
- **k** numeric(1) For speed, scores/P-values can be approximated after subsetting the motif every k columns when method = "exhaustive". If k is a value equal or higher to the size of input motif(s), then the calculations are exact. The default, 8, is recommended to those looking for a good tradeoff between speed and accuracy for jobs requiring repeated calculations. Note that this is ignored when method = "dynamic", as subsetting is not required.
- **nthreads** numeric(1) Run `motif_pvalue()` in parallel with nthreads threads. nthreads = 0 uses all available threads. Currently only applied when method = "exhaustive".
- **rand.tries** numeric(1) When ncol(motif) < k and method = "exhaustive", an approximation is used. This involves randomly approximating the overall motif score distribution. To increase accuracy, the distribution is approximated rand.tries times and the final scores averaged. Note that this is ignored when method = "dynamic", as subsetting is not required.
- **rng.seed** numeric(1) In order to allow `motif_pvalue()` to perform C++ level parallelisation, it must work independently from R. This means it cannot communicate with R to get/set the R RNG state. To get around this, the RNG seed used by the C++ function can be set with rng.seed. To make sure each thread gets a different seed however, the seed is multiplied with the iteration count. For example: when working with two motifs, the second motif gets the following seed: rng.seed * 2. The default is to pick a random number as chosen by `sample()`, which effectively makes `motif_pvalue()` dependent on the R RNG state. Note that this is ignored when method = "dynamic", as the random subsetting is only used for method = "exhaustive".
- **allow.nonfinite** logical(1) If FALSE, then apply a pseudocount if non-finite values are found in the PWM. Note that if the motif has a pseudocount greater than zero and the motif is not currently of type PWM, then this parameter has no effect as the pseudocount will be applied automatically when the motif is converted to a PWM internally. Note that this option is incompatible with method = "dynamic". A message will be printed if a pseudocount is applied. To disable this, set options(pseudocount.warning=FALSE).
- **method** character(1) One of c("dynamic", "exhaustive"). Algorithm used for calculating P-values. The "exhaustive" method involves finding all possible motif matches at or above the specified score using a branch-and-bound algorithm, which can be computationally intensive (Hartman et al., 2013). Additionally, the computation must be repeated for each hit. The "dynamic" method calculates the distribution of possible motif scores using a much faster dynamic programming algorithm, and can be recycled for multiple scores (Grant et al., 2011). The only disadvantage is the inability to use allow.nonfinite = TRUE.

**Details**

- **Regarding vectorization:**
A note regarding vectorizing the calculation when method = "dynamic" (no vectorization is possible with method = "exhaustive"): to avoid performing the P-value/score calculation repeatedly for individual motifs, provide the score/pvalue arguments as a list, with each entry corresponding to the scores/P-values to be calculated for the respective motifs provided to motifs. If you simply provide a list of repeating motifs and a single numeric vector of corresponding input scores/P-values, then `motif_pvalue()` will not vectorize. See the Examples section.

**The dynamic method:**

One of the algorithms available to `motif_pvalue()` to calculate scores or P-values is the dynamic programming algorithm used by FIMO (Grant et al., 2011). In this method, a small range of possible scores from the possible minimum and maximum is created and the cumulative probability of each score in this distribution is incrementally calculated using the logodds scores and the background probabilities. This distribution of scores and associated P-values can be used to calculate P-values or scores for any input, any number of times. This method scales well with large motifs, and multifreq representations. The only downside is that it is incompatible with `allow.nonfinite = TRUE`, as this would not allow for the creation of the initial range of scores. Although described for a different purpose, the basic premise of the dynamic programming algorithm is also described in Gupta et al. (2007).

**The exhaustive method:**

Calculating P-values exhaustively for motifs can be very computationally intensive. This is due to how P-values must be calculated: for a given score, all possible sequences which score equal or higher must be found, and the probability for each of these sequences (based on background probabilities) summed. For a DNA motif of length 10, the number of possible unique sequences is $4^{10} = 1,048,576$. Finding all possible sequences higher than a given score can be done very efficiently and quickly with a branch-and-bound algorithm, but as the motif length increases even this calculation becomes impractical. To get around this, the P-value calculation can be approximated.

In order to calculate P-values for longer motifs, this function uses the approximation proposed by Hartmann et al. (2013), where the motif is subset, P-values calculated for the subsets, and finally combined for a total P-value. The smaller the size of the subsets, the faster the calculation; but also, the bigger the approximation. This can be controlled by setting k. In fact, for smaller motifs (< 13 positions) calculating exact P-values can be done individually in reasonable time by setting k = 12.

To calculate a score from a P-value, all possible scores are calculated and the $(1 - pvalue) \times 100$ nth percentile score returned. When $k < ncol(motif)$, the complete set of scores is instead approximated by randomly adding up all possible scores from each subset. Note that this approximation can actually be potentially quite expensive at times and even slower than the exact version; for jobs requiring lots of repeat calculations, a bit of benchmarking beforehand can be useful to find the optimal settings.

Please note that bugs are more likely to occur when using the exhaustive method, as the algorithm contains several times more code compared to the dynamic method. Unless you have a strong need to use `allow.nonfinite = TRUE` then avoid using this method.

**Value**

numeric, list A vector or list of vectors of scores/P-values.
motif_pvalue

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

generate_motif, get_matches(), get_scores(), motif_range(), motif_score(), prob_match(), prob_match_bkg(), score_match()

Examples

```r
if (R.Version()$arch != "i386") {

## P-value/score calculations are performed using the PWM version of the
## motif
data(examplemotif)

## Get a minimum score based on a P-value
motif_pvalue(examplemotif, pvalue = 0.001)

## Get the probability of a particular sequence hit
motif_pvalue(examplemotif, score = 0)

## The calculations can be performed for multiple motifs
motif_pvalue(c(examplemotif, examplemotif), pvalue = c(0.001, 0.0001))

## Compare score thresholds and P-value:
scores <- motif_score(examplemotif, c(0.6, 0.7, 0.8, 0.9))
motif_pvalue(examplemotif, scores)

## Calculate the probability of getting a certain match or better:
TATATAT <- score_match(examplemotif, "TATATAT")
TATATAG <- score_match(examplemotif, "TATATAG")
motif_pvalue(examplemotif, TATATAT)
motif_pvalue(examplemotif, TATATAG)

## Get all possible matches by P-value:
get_matches(examplemotif, motif_pvalue(examplemotif, pvalue = 0.0001))

## Vectorize the calculation for multiple motifs and scores/P-values:
m <- create_motif()
motif_pvalue(c(examplemotif, m), list(1:5, 2:3))

## The non-vectorized equivalent:
```
motif_rc

Get the reverse complement of a DNA or RNA motif.

Description

For any motif, change the motif slot to its reverse complement. If the multifreq slot is filled, then it is also applied. No other slots are affected.

Usage

motif_rc(motifs, ignore.alphabet = FALSE)

Arguments

motifs See convert_motifs() for acceptable formats
ignore.alphabet logical(1) If TRUE, then motif_rc() throws an error when it detects a non-DNA/RNA motif. If FALSE, it will proceed regardless.

Value

See convert_motifs() for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

Examples

jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
    package = "universalmotif"))
jasper.rc <- motif_rc(jasper)
motif_tree

Generate ggplot2 motif trees with ggtree.

Description

For more powerful motif tree functions, see the motifStack package. The motif_tree() function compares motifs with compare_motifs() to create a distance matrix, which is used to generate a phylogeny. This can be plotted with ggtree::ggtree(). The purpose of this function is simply to combine the compare_motifs() and ggtree::ggtree() steps into one. For more control over tree creation, it is recommend to do these steps separately. See the "Motif comparisons and P-values" vignette for such a walkthrough. This function requires the ape and ggtree packages to be installed separately.

Usage

motif_tree(motifs, layout = "circular", linecol = "family",
           labels = "none", tipsize = "none", legend = TRUE,
           branch.length = "none", db.scores, method = "EUCL", use.type = "PPM",
           min.overlap = 6, min.position.ic = 0, tryRC = TRUE, min.mean.ic = 0,
           relative_entropy = FALSE, progress = FALSE, nthreads = 1,
           score.strat = "a.mean", ...)

Arguments

motifs list, dist See convert_motifs() for available formats. Alternatively, the resulting comparison matrix from compare_motifs() (run as dist(results)) beforehand; if the comparison was performed with a similarity metric, make sure to convert to distances first.

layout character(1) One of c('rectangular', 'slanted', 'fan', 'circular', 'radial', 'equal_angle', 'daylight'). See ggtree::ggtree().

linecol character(1) universalmotif slot to use to colour lines (e.g. 'family'). Not available for dist input (see examples for how to add it manually). See ggtree::ggtree().

labels character(1) universalmotif slot to use to label tips (e.g. 'name'). For dist input, only 'name' is available. See ggtree::ggtree().

tipsize character(1) universalmotif slot to use to control tip size (e.g. 'icscore'). Not available for dist input (see examples for how to add it manually). See ggtree::ggtree().

legend logical(1) Show legend for line colour and tip size. See ggtree::ggtree().

branch.length character(1) If 'none', draw a cladogram. See ggtree::ggtree().

db.scores data.frame See compare_motifs().

method character(1) One of PCC, EUCL, SW, KL, ALLR, Bhat, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.

use.type character(1)c('PPM', 'ICM'). The latter allows for taking into account the background frequencies (relative_entropy = TRUE). See compare_motifs().
**min.overlap** numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher than the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.

**min.position.ic** numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with `normalise.scores = TRUE`, as this will help punish scores resulting from only a fraction of an alignment.

**tryRC** logical(1) Try the reverse complement of the motifs as well, report the best score.

**min.mean.ic** numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use `average_ic()` to filter out low IC motifs to get around this if you want to avoid getting NAs in your output.

**relative_entropy** logical(1) Change the ICM calculation affecting `min.position.ic` and `min.mean.ic`. See `convert_type()`.

**progress** logical(1) Show message regarding current step.

**nthreads** numeric(1) Run `compare_motifs()` in parallel with `nthreads` threads. `nthreads = 0` uses all available threads.

**score.strat** character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

... **ggtree** params. See `ggtree::ggtree()`.

### Details
See `compare_motifs()` for more info on comparison parameters.

### Value
`ggplot` object.

### Author(s)
Benjamin Jean-Marie Tremblay, `<benjamin.tremblay@uwaterloo.ca>`

### References

**See Also**

`motifStack::motifStack()`, `compare_motifs()`, `ggtree::ggtree()`, `ggplot2::ggplot()`

**Examples**

```r
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt", package = "universalmotif"))
if (requireNamespace("ggtree", quietly = TRUE)) {
  jaspar.tree <- motif_tree(jaspar, linecol = "none", labels = "name",
                           layout = "rectangular")
}
## Not run:
## When inputting a dist object, the linecol and tipsize options are
## not available. To add these manually:
library(MotifDb)
library(ggtree)
library(ggplot2)

motifs <- filter_motifs(MotifDb, organism = "Athaliana")[1:50]
comparison <- compare_motifs(motifs, method = "PCC", score.strat = "a.mean")
comparison <- as.dist(1 - comparison)
mot.names <- attr(comparison, "Labels")
tree <- motif_tree(comparison)

annotations <- data.frame(label = mot.names,
                           icscore = sapply(motifs, function(x) x["icscore"]),
                           family = sapply(motifs, function(x) x["family"]))

tree <- tree %+% annotations +
  geom_tippoint(aes(size = icscore)) +
  aes(colour = family) +
  theme(legend.position = "right",
        legend.title = element_blank())
## End(Not run)
```

---

**Description**

read_cisbp

Usage

read_cisbp(file, skip = 0)

Arguments

file character(1) File name.
skip numeric(1) If not zero, will skip however many desired lines in the file before starting to read.

Details

CIS-BP motifs can be formatted with or without additional header metadata. Motifs without any header start at instances of the word "Pos", whereas motifs with a header start at instances of the word "TF".

Value

list universalmotif objects.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

Other read_motifs: read_homer(), read_jaspar(), read_matrix(), read_meme(), read_motifs(), read_transfac(), read_uniprobe()

Examples

cisbp <- read_cisbp(system.file("extdata", "cisbp.txt", package = "universalmotif"))
**read_homer**

Import HOMER motifs.

**Description**

Import HOMER formatted motifs. See [http://homer.ucsd.edu/homer/motif/](http://homer.ucsd.edu/homer/motif/). Assumed to be DNA motifs. Note that HOMER motifs come with a pre-determined logodds threshold; if you wish to re-create HOMER’s motif scanning, then use it in `scan_sequences()` (see examples).

**Usage**

```r
read_homer(file, skip = 0)
```

**Arguments**

- `file` character(1) File name.
- `skip` numeric(1) If not zero, will skip however many desired lines in the file before starting to read.

**Value**

list `universalmotif` objects.

**Author(s)**

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**References**


**See Also**

Other read_motifs: `read_cisbp()`, `read_jaspar()`, `read_matrix()`, `read_meme()`, `read_motifs()`, `read_transfac()`, `read_uniprobe()`

**Examples**

```r
data(ArabidopsisPromoters)
homer <- read_homer(system.file("extdata", "homer.txt", package = "universalmotif"))
thresholds <- homer |> to_df() |> with(logodds.threshold) |> as.numeric()
scan_sequences(homer, ArabidopsisPromoters, threshold = thresholds, threshold.type = "logodds.abs")
```
Import JASPAR motifs.

Description
Import JASPAR formatted motifs. See http://jaspar.genereg.net/. Can be either DNA, RNA, or AA motifs.

Usage
read_jaspar(file, skip = 0)

Arguments
file character(1) File name.
skip numeric(1) If not zero, will skip however many desired lines in the file before starting to read.

Value
list universalmotif objects.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References

See Also
Other read_motifs: read_cisbp(), read_homer(), read_matrix(), read_meme(), read_motifs(), read_transfac(), read_uniprobe()

Examples
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
package = "universalmotif")


read_matrix

Import motifs from raw matrices.

Description
Import simply formatted motifs.

Usage
read_matrix(file, skip = 0, type, positions = "columns", alphabet = "DNA", sep = "", headers = TRUE, rownames = FALSE, comment = NULL)

Arguments
- **file** character(1) File name.
- **skip** numeric(1) If not zero, will skip however many desired lines in the file before starting to read.
- **type** character(1) One of c(‘PCM’, ‘PPM’, ‘PWM’, ‘ICM’). If missing will try and guess which one.
- **positions** character(1) One of c(‘columns’, ‘rows’). Partial matching allowed. Indicate whether each position within a motif is represented as a row or a column in the file.
- **alphabet** character(1) One of c(‘DNA’, ‘RNA’, ‘AA’), or a string of letters.
- **sep** character(1) Indicates how individual motifs are separated. Set as NULL if there are no seperating lines between motifs (the default is to assume a blank line).
- **headers** logical(1), character(1) Indicating if and how to read names.
- **rownames** logical(1) Are there alphabet letters present as rownames?
- **comment** NULL, character(1) Character denoting lines to be considered comments.

Value
list universalmotif objects.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also
Other read_motifs: read_cisbp(), read_homer(), read_jaspar(), read_meme(), read_motifs(), read_transfac(), read_uniprobe()
Examples

```r
hocomoco <- system.file("exdata", "hocomoco.txt", package = "universalmotif")
hocomoco <- read_matrix(hocomoco, headers = ">", positions = "rows")
```

---

**read_meme**

Import MEME motifs.

Description

Import MEME formatted motifs, as well as original motif sequences. See [http://meme-suite.org/doc/meme-format.html](http://meme-suite.org/doc/meme-format.html). Both 'full' and 'minimal' formats are supported. DREME and STREME motifs can also be imported, but note that `readsites` and `readsites.meta` arguments do nothing.

Usage

```r
read_meme(file, skip = 0, readsites = FALSE, readsites.meta = FALSE,
          readsites.meta.tidy = FALSE)
```

Arguments

- `file` character(1) File name.
- `skip` numeric(1) If not zero, will skip however many desired lines in the file before starting to read.
- `readsites` logical(1) If TRUE, the motif sites will be read as well.
- `readsites.meta` logical(1) If `readsites = TRUE`, then additionally read site positions and P-values.
- `readsites.meta.tidy` logical(1) If `readsites.meta = TRUE`, merge the position site information for all motifs into a single tidy `data.frame`.

Details

Please note that the typical number precision limit in R is around 1e-308. This means that motif P-values in MEME files below this limit are rounded automatically to 0. To get around this, the E-value is also stored as a string in the `extrainfo` slot. If you require a numeric value for analysis, use the `log_string_pval()` function to get the log of the string-formatted p-value.

Value

- list `universalmotif` objects. If `readsites = TRUE`, a list comprising of a sub-list of motif objects and a sub-list of motif sites will be returned. If `readsites.meta = TRUE`, then two additional list items will be present, one containing site positions and P-values, and another containing combined sequence p-values. If `readsites.meta.tidy = TRUE`, an additional list entry named `sites.meta.tidy` will be added containing a `data.frame`. 
Import universalmotif formatted motifs.

Import motifs created from \texttt{write_motifs()}. For optimal storage of universalmotif class motifs, consider using \texttt{saveRDS()} and \texttt{readRDS()}. Currently the universalmotif format is YAML-based, but this is subject to change.

\textbf{Usage}

\texttt{read_motifs(file, skip = 0, progress = FALSE, BP = FALSE)}

\textbf{Arguments}

- **file**: character(1) File name.
- **skip**: numeric(1) If not zero, will skip however many desired lines in the file before starting to read.
- **progress**: logical(1) Show progress.
- **BP**: logical(1) Allows for the use of \texttt{BiocParallel} within \texttt{read_motifs()}. See \texttt{BiocParallel:::register()} to change the default backend.
read_transfac

Import TRANSFAC motifs.

Description

Import TRANSFAC formatted motifs. Assumed to be DNA motifs, type PCM. See `system.file("extdata", "transfac.txt", package="universalmotif")` for an example motif.

Usage

```r
read_transfac(file, skip = 0)
```

Arguments

- `file` character(1) File name.
- `skip` numeric(1) If not zero, will skip however many desired lines in the file before starting to read.

Details

A few TRANSFAC tags are recognized, including AC, ID, NA, HC and OS. HC will be set to the family slot and OS to the organism slot. If AC, ID and NA are present, then AC will be set as the motif name and NA as the alternate name. If AC is absent, then ID is set as the name. If ID is also absent, then NA is set as the motif name.

Value

- list `universalmotif` objects.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>
**References**


**See Also**

Other read_motifs: `read_cisbp()`, `read_homer()`, `read_jaspar()`, `read_matrix()`, `read_meme()`, `read_motifs()`, `read_uniprobe()`

**Examples**

```r
transfac <- read_transfac(system.file("extdata", "transfac.txt", package = "universalmotif"))
```

---

**Description**

Import UNIPROBE formatted motifs. Assumed DNA.

**Usage**

```r
read_uniprobe(file, skip = 0)
```

**Arguments**

- `file` character(1) File name.
- `skip` numeric(1) If not zero, will skip however many desired lines in the file before starting to read.

**Value**

list `universalmotif` objects.

**Author(s)**

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**References**

run_meme

See Also

Other read_motifs: read_cisbp(), read_homer(), read_jaspar(), read_matrix(), read_meme(), read_motifs(), read_transfac()

Examples

```r
uniprobe.minimal <- read_uniprobe(system.file("extdata", "uniprobe_minimal.txt", package = "universalmotif"))
uniprobe.full <- read_uniprobe(system.file("extdata", "uniprobe_full.txt", package = "universalmotif"))
```

run_meme

Run MEME from within R.

Description

De novo motif discovery via MEME. For a detailed description of the command, see http://meme-suite.org/doc/meme.html. For a brief description of the command parameters, call run_meme() without any arguments. Parameters in run_meme() which are directly taken from the MEME program are tagged with [MEME]. This function requires that the processx package be installed separately.

Usage

```r
run_meme(target.sequences, output = NULL, overwrite.dir = FALSE, control.sequences = NULL, weights = NULL, text = FALSE, brief = 1000, objfun = "classic", test = NULL, use.llr = FALSE, shuf = 2, hsfrac = NULL, cefrac = NULL, searchsize = NULL, norand = FALSE, csites = 1000, seed = 0, alph = NULL, revcomp = FALSE, pal = FALSE, mod = "zoops", nmotifs = 3, evt = NULL, nsites = NULL, minsites = NULL, maxsites = NULL, wnsites = 0.8, w = NULL, minw = 8, maxw = 50, allw = NULL, nomatrim = FALSE, wg = 11, ws = 1, noendgaps = FALSE, bfile = NULL, markov.order = 0, psp = NULL, maxiter = 50, distance = 0.001, prior = NULL, b = NULL)
```
plib = NULL, spfuzz = NULL, spmap = NULL, cons = NULL, p = NULL,
maxsize = NULL, maxtime = NULL, wd = getwd(), logfile = paste0(wd,
"/memerun.log"), readsites = TRUE, echo = FALSE, verbose = 1,
timeout = Inf, bin = getOption("meme.bin")

Arguments

target.sequences XStringSet List of sequences to get motifs from.

output character(1) Name of the output folder. If NULL, MEME output will be deleted.

overwrite.dir logical(1) If output is set but already exists, allow over-writing.

control.sequences XStringSet List of negative sequences. Only used if objfun = c("de", "se").

weights numeric Vector of numbers between 0 and 1, representing sequence weights.

text logical(1) [MEME]

brief numeric(1) [MEME]

objfun character(1) [MEME]

test character(1) [MEME]

use_llr logical(1) [MEME]

shuf numeric(1) [MEME]

hsfrac numeric(1) [MEME]

cefrac numeric(1) [MEME]

searchsize numeric(1) [MEME]

norand logical(1) [MEME]

csites numeric(1) [MEME]

seed numeric(1) [MEME]

alph character(1) [MEME] Note: custom alphabet definition files can be created using

meme_alph()

revcomp logical(1) [MEME]

pal logical(1) [MEME]

mod character(1) [MEME]

nmotifs numeric(1) [MEME]

evt numeric(1) [MEME]

nsites numeric(1) [MEME]

minsites numeric(1) [MEME]

maxsites numeric(1) [MEME]

wnsites numeric(1) [MEME]

w numeric(1) [MEME]

minw numeric(1) [MEME]
maxw numeric(1) [MEME]
allw numeric(1) [MEME]
nomatrim logical(1) [MEME]
wg numeric(1) [MEME]
ws numeric(1) [MEME]
noendgaps logical(1) [MEME]
bfile character(1) [MEME]
markov_order numeric(1) [MEME]
psp character(1) [MEME]
maxiter numeric(1) [MEME]
distance numeric(1) [MEME]
prior character(1) [MEME]
b numeric(1) [MEME]
plib character(1) [MEME]
spfuzz numeric(1) [MEME]
spmap character(1) [MEME]
cons character(1) [MEME]
p numeric(1) [MEME]
maxsize numeric(1) [MEME]
maxtime numeric(1) [MEME]
wd character(1) Working directory to run MEME in.
logfile character(1) File to dump MEME stderr. If NULL, no logs will be saved.
readsites logical(1) Read sites from MEME output (from read_meme()).
echo logical(1) Dump MEME output to console.
verbose numeric(1) Set verbose = 0 to quiet run_meme().
timeout numeric(1) Stop MEME program past timeout (seconds). See process::run().
bin character(1) Location of MEME binary. Alternatively, set this via options(meme.bin = '/path/to/meme/bin').

Value

list The output file is read with read_meme().

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References

**sample_sites**

Generate binding sites from a motif.

### Description

Given probabilities for a sequence as represented by a motif, generate random sequences with the same length as the motif.

### Usage

```r
sample_sites(motif, n = 100, use.freq = 1)
```

### Arguments

- **motif**: See `convert_motifs()` for acceptable formats.
- **n**: numeric(1) Number of sites to generate.
- **use.freq**: numeric(1) If one, use regular motif matrix. Otherwise, use respective multifreq matrix.

### Value

`XStringSet` object.

### Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

### See Also

`create_sequences()`, `create_motif()`, `add_multifreq()`

### Examples

```r
motif <- create_motif()
sites <- sample_sites(motif)
```
scan_sequences

Scan sequences for matches to input motifs.

Description

For sequences of any alphabet, scan them using the PWM matrices of a set of input motifs.

Usage

scan_sequences(motifs, sequences, threshold = 1e-04,
threshold.type = c("pvalue", "qvalue", "logodds", "logodds.abs"),
RC = FALSE, use.freq = 1, verbose = 0, nthreads = 1,
motif_pvalue.k = 8, use.gaps = TRUE, allow.nonfinite = FALSE,
warn.NA = TRUE, calc.pvals = TRUE, return.granges = FALSE,
no.overlaps = FALSE, no.overlaps.by.strand = FALSE,
no.overlaps.strat = c("score", "order"), respect.strand = FALSE,
motif_pvalue.method = c("dynamic", "exhaustive"),
calc.qvals = calc.pvals, calc.qvals.method = c("fdr", "BH",
"bonferroni"))

Arguments

motifs See convert_motifs() for acceptable motif formats.
sequences XStringSet Sequences to scan. Alphabet should match motif.
threshold numeric(1) See details.
threshold.type character(1) One of c("pvalue", "qvalue", "logodds", "logodds.abs"). See details.
RC logical(1) If TRUE, check reverse complement of the input sequences. Only available for DNA/RNA.
use.freq numeric(1) The default, 1, uses the motif matrix (from the motif["motif"] slot) to search for sequences. If a higher number is used, then the matching k-let matrix from the motif["multifreq"] slot is used. See add_multifreq().
verbose numeric(1) Describe progress, from none (0) to verbose (3).
nthreads numeric(1) Run scan_sequences() in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with only a single motif and sequence.
motif_pvalue.k numeric(1) Control motif_pvalue() approximation. See motif_pvalue(). Only used if motif_pvalue.method = "exhaustive".
use.gaps logical(1) Set this to FALSE to ignore motif gaps, if present.
allow.nonfinite logical(1) If FALSE, then apply a pseudocount if non-finite values are found in the PWM. Note that if the motif has a pseudocount greater than zero and the motif is not currently of type PWM, then this parameter has no effect as the pseudocount will be applied automatically when the motif is converted to a PWM.
Warn NA logical(1) Whether to warn about the presence of non-standard letters in the input sequence, such as those in masked sequences.

calc.pvals logical(1) Calculate P-values for each hit. This is a convenience option which simply gives motif_pvalue() the input motifs and the scores of each hit. Be careful about setting this to TRUE if you anticipate getting thousands of hits and are using motif_pvalue.method = "exhaustive": expect to wait a few seconds or minutes for the calculations to finish. Increasing the nthreads value can help greatly here. See Details for more information on P-value calculation. If motif_pvalue.method = "dynamic", then this is usually not an issue.

return.granges logical(1) Return the results as a GRanges object. Requires the GenomicRanges package to be installed.

no.overlaps logical(1) Remove overlapping hits from the same motifs. Overlapping hits from different motifs are preserved. Please note that the current implementation of this feature can add significantly to the run time for large inputs.

no.overlaps.by.strand logical(1) Whether to discard overlapping hits from the opposite strand (TRUE), or to only discard overlapping hits on the same strand (FALSE).

no.overlaps.strat character(1) One of c("score", "order"). The former option keeps the highest scoring overlapping hit (and the first of these within ties), and the latter simply keeps the first overlapping hit.

respect.strand logical(1) If motifs are DNA/RNA, then setting this option to TRUE will make scan_sequences() only scan the strands of the input sequences as indicated in the motif strand slot.

motif_pvalue.method character(1) One of c("dynamic", "exhaustive"). Algorithm used for calculating P-values. The "exhaustive" method involves finding all possible motif matches at or above the specified score using a branch-and-bound algorithm, which can be computationally intensive (Hartman et al., 2013). Additionally, the computation must be repeated for each hit. The "dynamic" method calculates the distribution of possible motif scores using a much faster dynamic programming algorithm, and can be recycled for multiple scores (Grant et al., 2011). The only disadvantage is the inability to use allow.nonfinite = TRUE. See motif_pvalue() for details.

calc.qvals logical(1) Whether to also calculate adjusted P-values. Only valid if calc.pvals = TRUE.

calc.qvals.method character(1) One of c("fdr", "BH", "bonferroni"). The method for calculating adjusted P-values. These are described in depth in the Sequence Searches vignette. Also see Noble (2009).
Details

Logodds scoring:
Similar to Biostrings::matchPWM(), the scanning method uses logodds scoring. (To see the scoring matrix for any motif, simply run convert_type(motif, "PWM"). For a multifreq scoring matrix: apply(motif["multifreq"], 2, ppm_to_pwm). In order to score a sequence, at each position within a sequence of length equal to the length of the motif, the scores for each base are summed. If the score sum is above the desired threshold, it is kept.

Thresholds:
If threshold.type = 'logodds', then the threshold value is multiplied by the maximum possible motif scores. To calculate the maximum possible scores a motif (of type PWM) manually, run motif_score(motif, 1). If threshold.type = 'pvalue', then threshold logodds scores are generated using motif_pvalue(). Finally, if threshold.type = 'logodds.abs', then the exact values provided will be used as thresholds. Finally, if threshold.type = 'pvalue', then the threshold is calculated as if threshold.type = 'pvalue' and the final set of hits are filtered based on their calculated Q-value. (Note: this means that the threshold score column will be incorrect!) This is done since most Q-values cannot be calculated prior to scanning. If you are running a very large job, it may be wise to use a P-value threshold followed by manually filtering by Q-value; this will avoid the scanning have to parse the larger number of hits from the internally-lowered threshold.

Non-standard letters:
Non-standard letters (such as "N", "+", ",", ",", etc in DNAString objects) will be safely ignored, resulting only in a warning and a very minor performance cost. This can used to scan masked sequences. See Biostrings::mask() for masking sequences (generating MaskedXString objects), and Biostrings::injectHardMask() to recover masked XStringSet objects for use with scan_sequences(). There is also a provided wrapper function which performs both steps: mask_seqs().

Value
DataFrame, GRanges with each row representing one hit. If the input sequences are DNAStringSet or RNAStringSet, then an additional column with the strand is included. Function args are stored in the metadata slot. If return.granges = TRUE then a GRanges object is returned.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References
sequence_complexity

Calculate sequence complexity.

Description

Calculate sequence complexity using either the Wootton-Federhen, Trifonov, or DUST algorithms.

Usage

sequence_complexity(seqs, window.size = 20,
                    window.overlap = round(window.size/2),
                    method = c("WoottonFederhen", "WoottonFederhenFast", "Trifonov", "TrifonovFast", "DUST"),
                    trifonov.max.word.size = 7, nthreads = 1, return.granges = FALSE)

Arguments

seqs XStringSet Input sequences.
window.size numeric Window size. If a number between 0 and 1 is provided, the value is calculated as the number multiplied by the sequence length.
window.overlap numeric Overlap between windows. If a number between 0 and 1 is provided, the value is calculated as the number multiplied by the sequence length.
method character(1) Choose one of the available methods for calculating sequence complexity. See details.

trifonov.max.word.size numeric(1) The maximum word size within each window used to calculate complexity using method = c("Trifonov", "TrifonovFast"). In other words, the Trifonov method involves counting the number of possible different sub-words in a window at different sizes up to the values provided by this option. It also involves calculating the product of ever increasing sequences of numbers and so in order to reduce the computations invoked this can be limited to a specific maximum sub-word size.

nthreads numeric(1) Run `sequence_complexity()` in parallel with nthreads threads. nthreads = 0 uses all available threads.

return.granges logical(1) Return the results as a GRanges object. Requires the GenomicRanges package to be installed.

Details

The Wootton-Federhen (Wootton and Federhen, 1993) and Trifonov (Trifonov, 1990) algorithms as well as their faster approximations are well described within Orlov and Potapov (2004). These algorithms score less complex sequences closer to 0, and more complex ones closer to 1. Please note that the ‘fast’ approximation versions of the two methods are not actually faster within `sequence_complexity()`, and so speed should not be a major consideration when choosing which method to use within the universalmotif package. The DUST algorithm implementation is described in Morgulis et al. (2006). In this case, less complex sequences score higher, and more complex ones closer to 0.

Briefly, the Wootton-Federhen complexity score is a reflection of the numbers of each unique letter found in the window (e.g. for DNA, the more of all four letters can be found in the window the higher the score). An increasing Trifonov score is a reflection of the numbers of increasingly larger k-mers (e.g. the count of possible 1-mers, 2-mers, 3-mers, ..., until trifonov.max.word.size). Finally, the DUST score approaches 0 as the count of unique 3-mers increases. (See the final section in the examples to see how different types of sequence compositions affect the methods.)

Please note that the authors of the different methods recommend various window sizes and complexity thresholds. The authors of DUST for example, suggest using a window size of 64 and a threshold of 2 for low complexity. Wootton and Federhen suggest a window size of 40, though show that 10 and 20 can be appropriate as well (for amino acid sequences). Keep in mind however that these algorithms were implemented at a time when computers were much slower; perhaps the authors would suggest different window sizes today. One thing to note is that the Wootton-Federhen algorithm has a hard limit due to the need to calculate the product from 1:window.size. This can end up calculating values which are greater than what a double can hold (e.g. try `prod(1:500)`). Its approximation does not suffer from this though, as it skips calculating the product.

In terms of speed, the Wootton-Federhen algorithms are fastest, with DUST being 1-3 times slower and the Trifonov algorithms being several times slower (though the exact amount depends on the max word size).

Value

DataFrame, GRanges with each row getting a complexity score for each window in each input sequence.
sequence_complexity

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

calc_complexity(), count_klets(), get_bkg(), mask_ranges(), mask_seqs()

Examples

## Feel free to play around with different toy sequences to get a feel for
## how the different methods perform

library(Biostrings)
test.seq <- DNAStringSet(c("AAAAAAAAAAA", "ATGACTGATGC"))

sequence_complexity(test.seq, method = "WoottonFederhen")
sequence_complexity(test.seq, method = "WoottonFederhenFast")
sequence_complexity(test.seq, method = "Trifonov")
sequence_complexity(test.seq, method = "TrifonovFast")
sequence_complexity(test.seq, method = "DUST")

## You could also use this in conjunction with mask_ranges() to hide
## low complexity regions from scanning, de novo motif discovery, etc

if (requireNamespace("GenomicRanges", quiet = TRUE)) {
  data(ArabidopsisPromoters)
  
  # Calculate complexity in 20 bp windows, sliding every 1 bp
  to.mask <- sequence_complexity(ArabidopsisPromoters, method = "DUST",
      window.size = 20, window.overlap = 19, return.granges = TRUE)
  
  # Get the ranges with a complexity score greater than 3.5
  to.mask <- to.mask[to.mask$complexity > 3.5]
  
  # See what the low complexity regions look like
  ArabidopsisPromoters[IRanges::reduce(to.mask)]

  # Mask them with the default '-' character:
  mask_ranges(ArabidopsisPromoters, to.mask)
To demonstrate how the methods work, consider:
These examples use the calc_complexity() utility which utilizes
the same algorithms and works on character vectors, but lacks
the ability to use sliding windows.)

```r
a <- "ACGT"

# For Wootton-Federhen, it can be easily shown it is only dependent
# on the counts of individual letters (though do note that the
# original paper discusses this method in the context of amino acid
calc_complexity("AAACCCGGTTT", alph = a) # 0.7707
calc_complexity("AACCGGTACGT", alph = a) # 0.7707
calc_complexity("ACGTACGTACGT", alph = a) # 0.7707

# As letters start to see drops in counts, the scores go down too:
calc_complexity("AAAACCCCGGGG", alph = a) # 0.6284
calc_complexity("AAAAACCCCCCC", alph = a) # 0.4105
calc_complexity("AAAAAAAAAACC", alph = a) # 0.2518

# Trifonov on the other hand is greatly affected by the number
# of higher order combinations:
calc_complexity("AAACCCGGTTT", c = "Trifonov", alph = a) # 0.6364
calc_complexity("AACCGGTACGT", c = "Trifonov", alph = a) # 0.7273

# This next one may seem surprising, but it indeed scores very low.
# This is because although it has many of each individual letter,
# the number of higher order letter combinations in fact is quite
# low due to this particular repeating pattern!
calc_complexity("ACGTACGTACGT", c = "Trifonov", alph = a) # 0.01231

# By extension, this means it scores sequences with fewer
# counts of individual letters lower too.
calc_complexity("AAAACCCCGGGG", c = "Trifonov", alph = a) # 0.2386
calc_complexity("AAAAACCCCCCC", c = "Trifonov", alph = a) # 0.0227
calc_complexity("AAAAAAAAAACC", c = "Trifonov", alph = a) # 0.0011

# As for DUST, it considers the number of 3-mers in the sequence.
# The higher the numbers of 3-mers, the lower the score.
# (0 = the max possible number of DNA 3-mers for the window size)
calc_complexity("AAACCCGGTTT", c = "DUST", alph = a) # 0
calc_complexity("AACCGGTACGT", c = "DUST", alph = a) # 0
calc_complexity("ACGTACGTACGT", c = "DUST", alph = a) # 0.8889
calc_complexity("AAAAACCCCCCC", c = "DUST", alph = a) # 0.333
calc_complexity("ACGACGACGACG", c = "DUST", alph = a) # 1.333
calc_complexity("AAAAACCCCCCC", c = "DUST", alph = a) # 1.333

# Similarly to Trifonov, the next one also scores as less complex
# compared to the previous one:
calc_complexity("ACACACACAC", c = "DUST", alph = a) # 2.222
calc_complexity("AAAAAACCCCCC", c = "DUST", alph = a) # 3.111
calc_complexity("AAAAAAAAAAAAAC", c = "DUST", alph = a) # 4
calc_complexity("AAAAAAAAAAAAAA", c = "DUST", alph = a) # 5
```
# Just to show once more why the seemingly more complex sequences
# such as "ACACACACACAC" score as less complex than "AAAAACCCCCC"
# for the Trifonov and DUST methods:
count_klets("ACACACACACAC", k = 3)  # Only 2 possible 3-mers
count_klets("AAAAACCCCCC", k = 3)  # Now 4 possible 3-mers!

## Description

Given a set of motifs, shuffle the columns to create new motifs. Currently does not support keeping the 'multifreq' slot. Only the 'bkg', 'nsites', 'strand', and 'bkgsites' slots will be preserved. Uses the same shuffling methods as `shuffle_sequences()`. When shuffling more than one motif, all motif columns are merged into a single pool and shuffled together, finally returning them as motifs of identical lengths as the input motifs. To instead shuffle motifs individually, call `shuffle_motifs()` using `lapply()`.

### Usage

```r
shuffle_motifs(motifs, k = 2, method = "linear")
```

### Arguments

- **motifs**: See `convert_motifs()` for acceptable formats.
- **k**: numeric(1) K-let size.
- **method**: character(1) Currently only 'linear' is accepted.

### Value

Motifs. See `convert_motifs()` for available output formats.

### Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

### See Also

- `shuffle_sequences()`
shuffle_sequences  Shuffle input sequences.

Description

Given a set of input sequences, shuffle the letters within those sequences with any k-let size.

Usage

shuffle_sequences(sequences, k = 1, method = "euler", nthreads = 1, 
    rng.seed = sample.int(10000, 1), window = FALSE, window.size = 0.1, 
    window.overlap = 0.01)

Arguments

sequences XStringSet Set of sequences to shuffle. Works with any set of characters.

k numeric(1) K-let size.

method character(1) One of c('euler', 'markov', 'linear'). Only relevant if k > 1. See details.

nthreads numeric(1) Run shuffle_sequences() in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with only a single sequence.

rng.seed numeric(1) Set random number generator seed. Since shuffling can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Each individual sequence in an XStringSet object will be given the following seed: rng.seed * index. The default is to pick a random number as chosen by sample(), which effectively is making shuffle_sequences() dependent on the R RNG state.

window logical(1) Shuffle sequences iteratively over windows instead of all at once.

window.size numeric(1) Window size. Can be a fraction less than one, or an integer representing the actual window size.

Details

markov method:
If method = 'markov', then the Markov model is used to generate sequences which will maintain (on average) the k-let frequencies. Please note that this method is not a 'true' shuffling, and for short sequences (e.g. <100bp) this can result in slightly more dissimilar sequences versus true shuffling. See Fitch (1983) for a discussion on the topic.
shuffle_sequences

**euler method:**
If `method = 'euler'`, then the sequence shuffling method proposed by Altschul and Erickson (1985) is used. As opposed to the 'markov' method, this one preserves exact k-let frequencies. This is done by creating a k-let edge graph, then following a random Eulerian walk through the graph. Not all walks will use up all available letters however, so the cycle-popping algorithm proposed by Propp and Wilson (1998) is used to find a random Eulerian path. A side effect of using this method is that the starting and ending sequence letters will remain unshuffled.

**linear method:**
If `method = 'linear'`, then the input sequences are split linearly every k letters. For example, for \( k = 3 \) 'ACAGATAGACCC' becomes 'ACA GAT AGA CCC'; after which these 3-lets are shuffled randomly.

**Single-letter shuffling:**
Do note however, that the `method` parameter is only relevant for \( k > 1 \). For \( k = 1 \), a simple shuffling is performed using the `shuffle` function from the C++ standard library.

**Value**

`XStringSet` The input sequences will be returned with identical names and lengths.

**Author(s)**
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**References**


**See Also**
`create_sequences()`, `scan_sequences()`, `enrich_motifs()`, `shuffle_motifs()`

**Examples**
```r
if (R.Version()$arch != "i386") {
sequences <- create_sequences()
sequences.shuffled <- shuffle_sequences(sequences, k = 2)
}
```
### switch_alph

*Switch between DNA and RNA alphabets.*

**Description**

Convert a motif from DNA to RNA, or RNA to DNA.

**Usage**

```r
switch_alph(motifs)
```

**Arguments**

- **motifs**
  
  See `convert_motifs()` for acceptable formats.

**Value**

The DNA/RNA version of the motifs. See `convert_motifs()` for acceptable output formats.

**Author(s)**

Benjamin Jean-Marie Tremblay, `<benjamin.tremblay@uwaterloo.ca>`

**See Also**

`create_motif()`

**Examples**

```r
DNA.motif <- create_motif()
RNA.motif <- switch_alph(DNA.motif)
```

### tidy-motifs

*Tidy manipulation of motifs.*

**Description**

Tidy manipulation of motifs.

**Usage**

```r
to_df(motifs, extrainfo = TRUE)
update_motifs(motif_df, extrainfo = TRUE, force = FALSE)
to_list(motif_df, extrainfo = TRUE, force = FALSE)
requires_update(motifs, extrainfo = TRUE)
```
Arguments

motifs List of motifs.
extrainfo Use the extrainfo slot in the tidy data.frame. The column names will be taken from the character vectors themselves, and unnamed elements will be assigned a unique name. To add elements to the slot, simply create new columns in the data.frame. Note that these will be coerced into characters. If extrainfo is not set to TRUE in to_df(), then the contents of the slot will not be transferred to the data.frame. If extrainfo is not set to TRUE in update_motifs() or to_list(), then the extra columns will be discarded.

motif_df Motif data.frame generated by to_df().
force Whether to coerce non-character data types into characters for inclusion in extrainfo. If force is FALSE (the default), columns which are not of type "character", "numeric", or "integer" (for example, list columns, or logical values), will not be added to the motif extrainfo slot, but will be passed onto the returned universalmotif_df unchanged. Setting force = TRUE coerces these values into a character, adding them to the extrainfo slot, and updating the universalmotif_df columns to reflect this coercion. In other words, forcing inclusion of these data is destructive and will change the column values. Use with caution.

Details

To turn off the informative messages/warnings when printing the object to the console, set options(universalmotif_df.warning=FALSE).

Value

For to_df(): a data.frame with the exposed slots as columns.
For update_motifs(): the updated data.frame.
For requires_update(): TRUE if the motifs are out of date, FALSE if otherwise. Note that this function uses identical() to check for this, which can be quite slow for large datasets. It is usually just as fast to simply run update_motifs() in such cases.
For to_list(): a list of motifs.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

Examples

## Not run:
library(universalmotif)
library(dplyr)

m <- c(create_motif(name = "motif A"), create_motif(name = "motif B"))

# Change the names of the motifs using the tidy way:
m <- m %>%
to_df() %>%
mutate(name = paste0(name, "-2")) %>%
trim_motifs

- **trim_motifs**

  Trim motifs.

**Description**

Remove edges of motifs with low information content. Currently does not trim multifreq representations.

**Usage**

```r
trim_motifs(motifs, min.ic = 0.25, trim.from = c("both", "left", "right"))
```

**Arguments**

- **motifs**
  - See [convert_motifs()](#) for acceptable formats.
- **min.ic**
  - numeric(1) Minimum allowed information content. See [convert_type()](#) for a discussion on information content.
- **trim.from**
  - character(1) Control the direction of trimming.

**Value**

Motifs See [convert_motifs()](#) for available output formats.

**Author(s)**

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**See Also**

- [create_motif()](#), [convert_type()](#)

**Examples**

```r
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt", package = "universalmotif"))
jasper.trimmed <- trim_motifs(jaspar)
```
universalmotif-class

universalmotif-class  universalmotif: Motif class.

Description

Container for motif objects. See create_motif() for creating motifs as well as a more detailed description of the slots. For a brief description of available methods, see examples.

Usage

## S4 method for signature 'universalmotif'
x[i]

## S4 replacement method for signature 'universalmotif'
x[i] <- value

## S4 method for signature 'universalmotif'
initialize(.Object, name, altname, family, organism,
motif, alphabet = "DNA", type, icscore, nsites, pseudocount = 1, bkg,
bkgsites, consensus, strand = "+-", pval, qval, eval, multifreq, extrainfo,
gapinfo)

## S4 method for signature 'universalmotif'
show(object)

## S4 method for signature 'universalmotif'
as.data.frame(x)

## S4 method for signature 'universalmotif'
subset(x, select)

## S4 method for signature 'universalmotif'
normalize(object)

## S4 method for signature 'universalmotif'
rowMeans(x)

## S4 method for signature 'universalmotif'
colMeans(x)

## S4 method for signature 'universalmotif'
colSums(x)

## S4 method for signature 'universalmotif'
rowSums(x)

## S4 method for signature 'universalmotif'

nrow(x)

## S4 method for signature 'universalmotif'
ncol(x)

## S4 method for signature 'universalmotif'
colnames(x)

## S4 method for signature 'universalmotif'
rownames(x)

cbind(..., deparse.level = 0)

Arguments

x  universalmotif Motif.
i  character Slot.
value  Object to replace slot with.
.Object  universalmotif Final motif.
name  character(1) Motif name.
altnames  character(1) Alternate motif name.
family  character(1) Transcription factor family.
organism  character(1) Species of origin.
motif  matrix Each column represents a position in the motif.
alphabet  character(1) One of c('DNA', 'RNA', 'AA'), or a combined string representing the letters.
type  character(1) One of c('PCM', 'PPM', 'PWM', 'ICM').
icscore  numeric(1) Total information content. Automatically generated.
nsites  numeric(1) Number of sites the motif was constructed from.
pseudocount  numeric(1) Correction to be applied to prevent ~Inf from appearing in PWM matrices.
bkg  numeric A vector of probabilities, each between 0 and 1. If higher order backgrounds are provided, then the elements of the vector must be named.
bkgsites  numeric(1) Total number of sites used to find the motif.
consensus  character(1) Consensus string. Automatically generated for 'DNA', 'RNA', and 'AA' alphabets.
strand  character(1) Whether the motif is specific to a certain strand.
pval  numeric(1) P-value associated with motif.
qval  numeric(1) Adjusted P-value associated with motif.
eval  numeric(1) E-value associated with motif.
multifreq  list See add_multifreq().
extrainfo character Any other extra information, represented as a named character vector.

gapinfo universalmotif_gapped(1) Gapped motif information.

object universalmotif Motif.

select numeric Columns to keep.

... universalmotif Motifs.

deparse.level Unused.

Value

A motif object of class universalmotif.

Slots

name character(1)
altname character(1)
family character(1)
organism character(1)
motif matrix
alphabet character(1)
type character(1)
iscore numeric(1) Generated automatically.

nsites numeric(1)
pseudocount numeric(1)
bkg numeric 0-order probabilities must be provided for all letters.
bkgsites numeric(1)
consensus character Generated automatically.
strand character(1)
pval numeric(1)
qval numeric(1)
eval numeric(1)
multifreq list
extrainfo character
gapinfo universalmotif_gapped(1)

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>
Examples

```r
## Access the slots.
motif <- create_motif()
motif["motif"]
# you can also access multiple slots at once, released as a list
motif["motif", "name"]

## Replace the slots.
motif["name"] <- "new name"
# some slots are protected
# motif["consensus"] <- "AAAA" # not allowed

## Assemble a list of motifs.
c(motif, motif)

## Represent a motif as a data.frame. The actual motif matrix is lost.
## Necessary for `summarise_motifs`.
as.data.frame(motif)

## Subset a motif matrix by column.
subset(motif, 3:7) # extract motif core

## Apply the pseudocount slot (or `1`, if the slot is set to zero) to the
## motif matrix.
motif2 <- create_motif("AAAA", nsites = 100, pseudocount = 1)
normalize(motif2)

## Calculate motif rowMeans.
rowMeans(motif)

## Calculate motif colMeans.
colMeans(motif)

## Calculate motif colSums
colSums(motif)

## Calculate motif rowSums.
rowSums(motif)

## Count motif rows.
nrow(motif)
```
### ncol

Count motif columns.

```r
ncol(motif)
```

### colnames

Get motif colnames.

```r
colnames(motif)
```

### rownames

Get motif rownames.

```r
rownames(motif)
```

### cbind

Bind motifs together to create a new motif.

```r
cbind(motif, motif2)
```

---

**universalmotif-pkg**  
**universalmotif:** Import, Modify and Export Motifs with R

---

**Description**

A collection of utility functions for working with motifs.

---

**utilities**  
**Utility functions.**

---

**Description**

Utility functions have been split into two categories: those related to motifs ?’utils-motif’, and those related to sequences ?’utils-sequence’.

**Author(s)**

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**See Also**

`utils-motif`, `utils-sequence`
utils-motif

Motif-related utility functions.

Description

Motif-related utility functions.

Usage

add_gap(motif, gaploc = ncol(motif)%/%2, mingap = 1, maxgap = 5)

average_ic(motifs, average = c("a.mean", "g.mean", "median", "fzt"))

compare_columns(x, y, method, bkg1 = rep(1/length(x), length(x)),
    bkg2 = rep(1/length(y), length(y)), nsites1 = 100, nsites2 = 100)

consensus_to_ppm(letter)

consensus_to_ppmAA(letter)

get_consensus(position, alphabet = "DNA", type = "PPM", pseudocount = 1)

get_consensusAA(position, type = "PPM", pseudocount = 0)

get_matches(motif, score, allow.nonfinite = FALSE)

get_scores(motif, allow.nonfinite = FALSE)

icm_to_ppm(position)

motif_range(motif, use.freq = 1, allow.nonfinite = FALSE)

motif_score(motif, threshold = c(0, 1), use.freq = 1,
    allow.nonfinite = FALSE, threshold.type = c("total", "fromzero"))

log_string_pval(pval)

pcm_to_ppm(position, pseudocount = 0)

position_icscore(position, bkg = numeric(), type = "PPM",
    pseudocount = 1, nsites = 100, relative_entropy = FALSE,
    schneider_correction = FALSE)

ppm_to_icm(position, bkg = numeric(), schneider_correction = FALSE,
    nsites = 100, relative_entropy = FALSE)

ppm_to_pcm(position, nsites = 100)
utils-motif

ppm_to_pwm(position, bkg = numeric(), pseudocount = 1, nsites = 100, smooth = TRUE)

prob_match(motif, match, allow.zero = TRUE)

prob_match_bkg(bkg, match)

pwm_to_ppm(position, bkg = numeric())

round_motif(motif, pct.tolerance = 0.05)

score_match(motif, match, allow.nonfinite = FALSE)

summarise_motifs(motifs, na.rm = TRUE)

ungap(motif, delete = FALSE)

Arguments

**motif**
Motif object to calculate scores from, or add/remove gap, or round.

**gaploc**
numeric Motif gap locations. The gap occurs immediately after every position value. If missing, uses \round{ncol(motif) / 2}.

**mingap**
numeric Minimum gap size. Must have one value for every location. If missing, set to 1.

**maxgap**
numeric Maximum gap size. Must have one value for every location. If missing, set to 5.

**motifs**
list A list of universalmotif motifs.

**average**
character(1) One of c("a.mean", "g.mean", "median", "fzt"). How to calculate the average motif information content.

**x**
numeric First column for comparison.

**y**
numeric Second column for comparison.

**method**
character(1) Column comparison metric. See `compare_motifs()` for details.

**bkg1**
numeric Vector of background probabilities for the first column. Only relevant if method = "ALLR".

**bkg2**
numeric Vector of background probabilities for the second column. Only relevant if method = "ALLR".

**nsites1**
numeric(1) Number of sites for the first column. Only relevant if method = "ALLR".

**nsites2**
numeric(1) Number of sites for the second column. Only relevant if method = "ALLR".

**letter**
character(1) Any DNA, RNA, or AA IUPAC letter. Ambiguity letters are accepted.

**position**
numeric A numeric vector representing the frequency or probability for each alphabet letter at a specific position.
alphabet 
character(1) One of c('DNA', 'RNA').

type 
character(1) One of c('PCM', 'PPM', 'PWM', 'ICM').
pseudocount 
umERIC(1) Used to prevent zeroes in motif matrix.
score 
umERIC(1) Logodds motif score.
allow.nonfinite 
logical(1) If FALSE, then apply a pseudocount if non-finite values are found in the PWM. Note that if the motif has a pseudocount greater than zero and the motif is not currently of type PWM, then this parameter has no effect as the pseudocount will be applied automatically when the motif is converted to a PWM internally. This value is set to FALSE by default in order to stay consistent with pre-version 1.8.0 behaviour. A message will be printed if a pseudocount is applied. To disable this, set options(pseudocount.warning=FALSE).

use.freq 
umERIC(1) Use regular motif or the respective multifreq representation.
threshold 
umERIC(1) Any number of numeric values between 0 and 1 representing score percentage.
threshold.type 
character For "total", a threshold of zero represents the minimum possible score. This means the range of scores that can be extracted is from the minimum to the maximum possible scores. For "fromzero", a threshold of zero is a score of zero. This means the range of scores is from zero to the maximum. The "total" threshold type can only be used if no non-finite values are present in the PWM.

pval 
character(1) String-formatted p-value.
bkg 
umERIC Should be the same length as the alphabet length.
nsites 
umERIC(1) Number of sites motif originated from.
relative_entropy 
logical(1) Calculate information content as relative entropy or Kullback-Leibler divergence.
schneider_correction 
logical(1) Apply sample size correction.
smooth 
logical(1) Apply pseudocount correction.
match 
character Sequence string to calculate score from.
allow.zero 
logical(1) If FALSE, apply a pseudocount if zero values are found in the background frequencies.
pct.tolerance 
numeric(1) or character(1) The minimum tolerated proportion each letter must represent per position in order not to be rounded off, either as a numeric value from 0 to 1 or a percentage written as a string from "0%" to "100%".

na.rm 
logical Remove columns where all values are NA.
delete 
logical(1) Clear gap information from motif. If FALSE, then it can be reactivated simply with add_gap(motif).
Value

For `consensus_to_ppm()` and `consensus_to_ppmAA()`: a numeric vector of length 4 and 20, respectively.
For `get_consensus()` and `get_consensusAA()`: a character vector of length 1.
For `get_matches()`: a character vector of motif matches.
For `motif_range()`: a named numeric vector of motif scores.
For `motif_score()`: a named numeric vector of motif scores.
For `log_string_pval()`: a numeric vector of length 1.
For `position_icscore()`: a numeric vector of length 1.
For `ppm_to_icm()`, `icm_to_ppm()`, `pcm_to_ppm()`, `ppm_to_pcm()`, `ppm_to_pwm()`, and `pwm_to_ppm()`:
a numeric vector with length equal to input numeric vector.
For `prob_match()`: a numeric vector of probabilities.
For `round_motif()`: the input motif, rounded.
For `score_match()`: a numeric vector with the match motif score.
For `summarise_motifs()`: a data.frame with columns representing the `universal_motif` slots.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

`create_motif()`

Examples

data(examplemotif)
examplerotif0 <- examplemotif
eexamplemotif0["pseudocount"] <- 0

##########################################################################
## add_gap
## Add gap information to a motif.
m <- create_motif()
# Add a gap size 5-8 between positions 4 and 5:
m <- add_gap(m, gaploc = 4, mingap = 5, maxgap = 8)

##########################################################################
## average_ic
## Calculate the average information content for a list of motifs.
m <- create_motif()
average_ic(m, "fzt")

##########################################################################
## compare_columns
## Compare two numeric vectors using the metrics from compare_motifs()
compare_columns(c(0.5, 0.1, 0.1, 0.2), c(0.7, 0.1, 0.1, 0.1), "PCC")
## consensus_to_ppm
Do the opposite of get_consensus. Note that loss of information is inevitable. Generates a sequence matrix.
sapply(c("A", "G", "T", "B"), consensus_to_ppm)

## consensus_to_ppmAA
Do the opposite of get_consensusAA and generate a motif matrix.
sapply(c("V", "A", "L"), consensus_to_ppmAA)

## get_consensus
Get a consensus string from a DNA/RNA motif.
m <- create_motif()["motif"]
apply(m, 2, get_consensus)

## get_consensusAA
Get a consensus string from an amino acid motif. Unless each position is clearly dominated by a single amino acid, the resulting string will likely be useless.
m <- create_motif(alphabet = "AA")["motif"]
apply(m, 2, get_consensusAA, type = "PPM")

## get_match
Get all possible motif matches above input score
get_matches(examplemotif, 0)
get_matches(examplemotif0, 0, allow.nonfinite = TRUE)

## get_scores
Get all possible scores for a motif
length(get_scores(examplemotif))
get_scores(examplemotif)
get_scores(examplemotif0, allow.nonfinite = TRUE)

## icm_to_ppm
Do the opposite of ppm_to_icm.
m <- create_motif(type = "ICM")["motif"]
apply(m, 2, icm_to_ppm)

## motif_range
Calculate the range of possible logodds scores for a motif
motif_range(examplemotif)
motif_range(examplemotif, allow.nonfinite = TRUE)

## motif_score
## Calculate motif score from different thresholds
m <- normalize(examplemotif)
motif.score(m, c(0, 0.8, 1))
motif.score(examplemotif, c(0, 0.8, 1), allow.nonfinite = TRUE,
    threshold.type = "fromzero")

### log_string_pval
### Get the log of a string-formatted p-value
log_string_pval("1e-200")

### pcm_to_ppm
### Go from a count type motif to a probability type motif.
m <- create_motif(type = "PCM", nsites = 50)["motif"]
apply(m, 2, pcm_to_ppm, pseudocount = 1)

### position_icscore
### Similar to ppm_to_icm, except this calculates the position sum.
m <- create_motif()"motif"]
apply(m, 2, position_icscore, type = "PPM", bkg = rep(0.25, 4))

### ppm_to_icm
### Convert one column from a probability type motif to an information
### content type motif.
m <- create_motif(nsites = 100, pseudocount = 0.8)["motif"]
apply(m, 2, ppm_to_icm, nsites = 100, bkg = rep(0.25, 4))

### ppm_to_pcm
### Do the opposite of pcm_to_ppm.
m <- create_motif()"motif"]
apply(m, 2, ppm_to_pcm, nsites = 50)

### ppm_to_pwm
### Go from a probability type motif to a weight type motif.
m <- create_motif()"motif"]
apply(m, 2, ppm_to_pwm, nsites = 100, bkg = rep(0.25, 4))

### prob_match, prob_match_bkg
### Calculate probability of a particular match based on background
### frequencies
prob_match(examplemotif, "TATATAT")
### Since this motif has a uniform background, the probability of
### finding any motif hit within the sequence is equal
prob_match(examplemotif, "TATATAG")
m <- examplemotif
m["bkg"] <- c(0.3, 0.2, 0.2, 0.3)
prob_match(m, "TATATAT")
## The prob_match_bkg alternative allows you to simply pass along the background frequencies

```r
prob_match_bkg(c(A=0.3, C=0.2, G=0.2, T=0.3), c("TATATAT", "TATATAG"))
```

#### pwm_to_ppm

```r
# Do the opposite of ppm_to_pwm.
m <- create_motif(type = "PWM")[["motif"]]
apply(m, 2, pwm_to_ppm, bkg = rep(0.25, 4))
```

#### round_motif

```r
# Round down letter scores to 0
m <- create_motif()
# Remove letters from positions which are less than 5% of the total position:
round_motif(m, pct.tolerance = 0.05)
```

#### score_match

```r
# Calculate score of a particular match
score_match(examplemotif, "TATATAT")
score_match(examplemotif, "TATATAG")
score_match(examplemotif0, "TATATAT", allow.nonfinite = TRUE)
score_match(examplemotif0, "TATATAG", allow.nonfinite = TRUE)
```

#### summarise_motifs

```r
# Create a data.frame of information based on a list of motifs.
m1 <- create_motif()
m2 <- create_motif()
m3 <- create_motif()
summarise_motifs(list(m1, m2, m3))
```

#### ungap

```r
# Unset motif's gap status. Does not delete actual gap data unless delete = TRUE.
m <- create_motif()
m <- add_gap(m, 3, 2, 4)
m <- ungap(m)
# Restore gap data:
m <- add_gap(m)
```
Description

Sequence-related utility functions.

Usage

```r
calc_complexity(string, complexity.method = c("WoottonFederhen", "WoottonFederhenFast", "Trifonov", "TrifonovFast", "DUST"), alph = NULL, trifonov.max.word.size = 7)
```

```r
calc_windows(n, window = 1, overlap = 0, return.incomp = TRUE)
```

```r
count_klets(string, k = 1, alph)
```

```r
get_klets(lets, k = 1)
```

```r
mask_ranges(seqs, ranges, letter = "-")
```

```r
mask_seqs(seqs, pattern, RC = FALSE, letter = "-")
```

```r
meme_alph(core, file = stdout(), complements = NULL, ambiguity = NULL, like = NULL, alph.name = NULL, letter.names = NULL, colours = NULL)
```

```r
shuffle_string(string, k = 1, method = c("euler", "linear", "markov"), rng.seed = sample.int(10000, 1))
```

```r
slide_fun(string, FUN, FUN.VALUE, window = 1, overlap = 0, return.incomp = TRUE)
```

```r
window_string(string, window = 1, overlap = 0, return.incomp = TRUE, nthreads = 1)
```

Arguments

- `string` character(1) A character vector containing a single string, with the exception of `calc_complexity()` where string can be a length greater than one.
- `complexity.method` character(1) Complexity algorithm. See `sequence_complexity()`.
- `alph` character(1) A single character string with the desired sequence alphabet. If missing, finds the unique letters within each string.
- `trifonov.max.word.size` integer(1) Maximum word size for use in the Trifonov complexity methods. See `sequence_complexity()`.
- `n` integer(1) Total size from which to calculate sliding windows.
- `window` integer(1) Window size to slide along.
- `overlap` integer(1) Overlap size between windows.
- `return.incomp` logical(1) Whether to return the last window if it is smaller then the requested window size.
utils-sequence

k  
integer(1) K-let size.

lets  
character A character vector where each element will be considered a single unit.

seqs  
XStringSet Sequences to mask. Cannot be BStringSet.

ranges  
GRanges The ranges to mask. Must be a GRanges object from the GenomicRanges package.

letter  
character(1) Character to use for masking.

pattern  
character(1) Pattern to mask.

RC  
logical(1) Whether to mask the reverse complement of the pattern.

core  
character(1) Core alphabet symbols. If complements are also provided, then only half of the letters should be provided to this argument.

file  
Output file.

complements  
character(1), NULL Complementary letters to the core symbols.

ambiguity  
character(1), NULL A named vector providing ambiguity codes for the custom alphabet.

like  
character(1), NULL How to classify the custom alphabet. If not NULL, then one of c("DNA", "RNA", "PROTEIN").

alph.name  
character(1), NULL Custom alphabet name.

letter.names  
character, NULL Named vector of core symbol names.

colours  
character, NULL Named vector of core symbol colours. MEME requires hex colours.

method  
character(1) Shuffling method. One of c("euler", "linear", "markov"). See shuffle_sequences().

rng.seed  
numeric(1) Set random number generator seed. Since shuffling in shuffle_sequences() can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Since shuffle_string() uses the same underlying code as shuffle_sequences(), it also requires a separate seed even if it is run in serial.

FUN  
closure The function to apply per window. (See ?vapply.)

FUN.VALUE  
The expected return type for FUN. (See ?vapply.)

nthreads  
integer(1) Number of threads to use. Zero uses all available threads.

Value

For calc_complexity(): A vector of numeric values.
For calc_windows(): A data.frame with columns start and stop.
For count_klets(): A data.frame with columns lets and counts.
For get_klets(): A character vector of k-lets.
For mask_ranges(): The masked XStringSet object.
For mask_seqs(): The masked XStringSet object.
For meme_alph(): NULL, invisibly.
For **shuffle_string()**: A single character string.
For **slide_fun()**: A vector with type **FUN.VALUE**.
For **window_string()**: A character vector.

**Author(s)**
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**See Also**
create_sequences(), get_bkg(), sequence_complexity(), shuffle_sequences()

**Examples**
```r
# calc_complexity
# Calculate complexity for arbitrary strings
calc_complexity("GTGCCCGCGGAACCCCGC", c = "WoottonFederhen")
calc_complexity("GTGCCCGCGGAACCCCGC", c = "WoottonFederhenFast")
calc_complexity("GTGCCCGCGGAACCCCGC", c = "Trifonov")
calc_complexity("GTGCCCGCGGAACCCCGC", c = "TrifonovFast")
calc_complexity("GTGCCCGCGGAACCCCGC", c = "DUST")
```

```r
# calc_windows
# Calculate window coordinates for any value 'n'.
calc_windows(100, 10, 5)
```

```r
# count_klets
# Count k-lets for any string of characters
count_klets("GCAAATGTACGCAGGGCCGA", k = 2)
# The default 'k' value (1) counts individual letters
count_klets("GCAAATGTACGCAGGGCCGA")
```

```r
# get_klets
# Generate all possible k-lets for a set of characters
get_klets(c("A", "C", "G", "T"), 3)
# Note that each element in 'lets' is considered a single unit;
# see:
get_klets(c("AA", "B"), k = 2)
```

```r
# mask_ranges
# Mask arbitrary ranges
if (requireNamespace("GenomicRanges", quiet = TRUE)) {
  ranges <- GenomicRanges::GRanges("A", IRanges::IRanges(1, 5))
  seq <- Biostats::DNAStringSet(c(A = "ATGACTGATTCTATA"))
  mask_ranges(seq, ranges, "-")
}
```
# view_logo

Plot logos from numeric matrices.

## Description

This function provides the plotting capabilities of `view_motifs()` without requiring `universalmotif-class` objects. Instead, it takes a numeric matrix with row names as input. Additionally, columns can be of any height and letters can be a mix of different character lengths.

## Usage

```r
view_logo(x, fontDF = NULL, fill = "black", colour.scheme = NULL, min.height = 0.01, x.spacer = 0.04, y.spacer = 0.01, sort.positions = FALSE, sort.positions.decreasing = TRUE, fit.to.height = NULL)
```
Arguments

x
A numeric matrix with row names. The row names can be a mix of different character lengths.

fontDF
data.frame or DataFrame Polygon data for letters used for plotting, as generated by the createPolygons() function from the gglogo package. See the fontDFroboto data object (which is used by default when fontDF = NULL). See Examples for how to generate your own font set. Expected columns: x, y, order, group; additional columns will be ignored.

fill
character A single colour to fill all letters with. Ignored if colour.scheme is provided.

colour.scheme
character A named character vector of colour names. Provide colours for individual letters, even if the row names are made up of multiple characters.

min.height
numeric(1) Minimum height for a letter to be plotted. The number is taken as the fraction of the total height of the plot. The default value is to not show letters which take up 1% or less of the vertical space. For smaller figures it is recommended to increase this value, and vice versa for larger figures.

x.spacer
numeric(1) Add horizontal spacing between letters. The number is taken as the fraction of the width of an individual position. Increasing this value is recommended for letters made up of multiple characters.

y.spacer
numeric(1) Add vertical spacing between letters. The number is taken as the fraction of the total height of the plot.

sort.positions
logical(1) Sort letters vertically per position by height.

sort.positions.decreasing
logical(1) Sort in decreasing or increasing order based on letter height.

fit.to.height
numeric(1) Normalize the per position height to this value. If NULL, no normalization is applied. Note that this parameter is ignored if use.type = c("PWM", "ICM").

Value

A ggplot object. If you wish to plot the data yourself from polygon paths, access them using $data on the output object. The theme theme_void() is applied to the object; apply your own theme or adjust specific plot parameters with theme() to change this.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

view_motifs()
## view_motifs

### Examples

```r
## Feel free to mix and match row name character lengths and column sums.
data(examplemotif)
toplot <- examplemotif[, "motif"]
toplot[20] <- -0.5
rownames(toplot)[1] <- "AA"
view_logo(toplot)
```

### Description

Show sequence logo. If given a list of more than one motif, then the motifs are aligned with the first in the list.

### Usage

```r
view_motifs(motifs, use.type = "ICM", method = "ALLR", tryRC = TRUE,
min.overlap = 6, min.mean.ic = 0.25, relative_entropy = FALSE,
normalise.scores = FALSE, min.position.ic = 0, score.strat = "sum",
return.raw = FALSE, dedup.names = TRUE, show.positions = TRUE,
show.positions.once = TRUE, show.names = TRUE, names.pos = c("top",
"right"), use.freq = 1, colour.scheme = NULL, fontDF = NULL,
min.height = 0.01, x.spacer = if (use.freq == 1) 0.04 else 0.1,
y.spacer = 0.01, sort.positions = !use.type %in% c("PCM", "PPM"),
sort.positions.decreasing = TRUE, text.size = 16, fit.to.height = if
(use.type == "PPM") 1 else NULL, RC.text = "[RC]", ...)
```

### Arguments

- **motifs**: See `convert_motifs()` for acceptable motif formats.
- **use.type**: character(1) One of c('PCM', 'PPM', 'PWM', 'ICM').
- **method**: character(1) One of PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
- **tryRC**: logical(1) Try the reverse complement of the motifs as well, report the best score.
- **min.overlap**: numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
min.mean.ic numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs. Note that this can result in some comparisons failing if no alignment passes the mean IC threshold. Use average_ic() to filter out low IC motifs to get around this if you want to avoid getting NAs in your output.

relative_entropy logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic. See convert_type().

normalise.scores logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.

min.position.ic numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.

score.strat character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

return.raw logical(1) Instead of returning a plot, return the aligned named matrices used to generate the plot. This can be useful if you wish to use view_motifs() alignment capabilities for custom plotting uses. Alignment is performed by adding empty columns to the left or right of motifs to generate matrices of equal length.

dedup.names logical(1) Plotting motifs with duplicated names is not allowed. Setting this to TRUE allows the names to be modified for plotting.

show.positions logical(1) Show x-axis position tick labels.

show.positions.once logical(1) When plotting multiple motifs, show x-axis position tick labels only once. If FALSE, then x-axis tick labels are specific to each motif.

show.names logical(1) Add motif names when plotting multiple motifs.

names.pos character(1) Motif name locations. Either above (top) or to the right (right) of the logos.

use.freq numeric(1) Plot higher order motifs from the multifreq slot.

colour.scheme character A named character vector of colour names. Default colours are provided for DNA, RNA, and AA motifs if left NULL.

fontDF data.frame or DataFrame Polygon data for letters used for plotting, as generated by the createPolygons() function from the gglogo package. See the fontDFroboto data object (which is used by default when fontDF = NULL). See Examples for how to generate your own font set. Expected columns: x, y, order, group; additional columns will be ignored.
view_motifs

min.height numeric(1) Minimum height for a letter to be plotted. The number is taken as the fraction of the total height of the plot. The default value is to not show letters which take up 1% or less of the vertical space. For smaller figures it is recommended to increase this value, and vice versa for larger figures.

x.spacer numeric(1) Add horizontal spacing between letters. The number is taken as the fraction of the width of an individual position. Increasing this value is recommended for plotting multifreq motifs.

y.spacer numeric(1) Add vertical spacing between letters. The number is taken as the fraction of the total height of the plot.

sort.positions logical(1) Sort letters vertically per position by height.

sort.positions.decreasing logical(1) Sort in decreasing or increasing order based on letter height.

text.size numeric(1) Text size for labels.

fit.to.height numeric(1) Normalize the per position height to this value. If NULL, no normalization is applied. Note that this parameter is ignored if use.type = c("PWM", "ICM").

RC.text character(1) The text to display alongside the name of motifs shown as their reverse complement.

... Unused. Was previously in place to allow extra args to be given to ggseqlogo::ggseqlogo, however universalmotif now implements its own motif plotting code directly with ggplot2.

Details

See compare_motifs() for more info on comparison parameters.

See view_logo() to plot from a numeric matrix with arbitrary values instead of a motif object.

Note: score.strat = "a.mean" is NOT recommended, as view_motifs() will not discriminate between two alignments with equal mean scores, even if one alignment is longer than the other.

Note: if you want to plot the motifs yourself, you can set return.raw=TRUE to get the numeric motif matrices and calculate the polygon paths on your own or access the polygon path data directly from the final ggplot object using $data.

Value

A ggplot object. If return.raw = TRUE, a list of matrices.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

compare_motifs(), add_multifreq(), view_logo()
write_homer

Examples

```r
## Plotting multifreq motifs:
data(examplemotif2)
view_motifs(examplemotif2, use.freq = 2)

## Generate your own letter set:
## Not run:
library(gglogo)  # install from CRAN first if needed
fontDFtimes <- createPolygons(LETTERS, "Times", 800, scale = TRUE)
view_motifs(examplemotif2, fontDF = fontDFtimes)

## Note: setting `scale = TRUE` is necessary to properly align letters
## vertically, but this has the effect of horizontally stretching out
## letters which shouldn’t be stretched (such as "I"). If you need to plot
## letters which have been badly horizontally scaled, you can fix them
## manually as demonstrated here:

# Retrieve the x-coordinates for the desired letter:
tofix <- fontDFtimes$x[fontDFtimes$group == "I"]
# Scale the letter x-coordinates:
tofix <- tofix * 0.35
# Remember to center the letter around 0.5 again:
tofix <- tofix + (1 - max(tofix)) / 2
# Apply the fix:
fontDFtimes$x[fontDFtimes$group == "I"] <- tofix
view_motifs(create_motif("AIG", alphabet = "AA"), fontDF = fontDFtimes)

## End(Not run)
```

write_homer

Export motifs in HOMER format.

Description

Convert DNA motifs to HOMER format and write to file. See [http://homer.ucsd.edu/homer/motif/](http://homer.ucsd.edu/homer/motif/).

Usage

```r
write_homer(motifs, file, logodds_threshold = NULL, overwrite = FALSE,
append = FALSE, threshold = 0.8, threshold.type = c("logodds",
"logodds.abs", "pvalue"))
```
**Arguments**

- **motifs**
  - See `convert_motifs()` for acceptable formats.

- **file**
  - character(1) File name.

- **logodds_threshold**
  - Deprecated. If set, `read_homer()` will behave like pre-version 1.12.0 of the universalmotif package for backwards compatibility (though a warning will be printed).

- **overwrite**
  - logical(1) Overwrite existing file.

- **append**
  - logical(1) Add to an existing file.

- **threshold**
  - numeric(1) Stringency required for HOMER to match a motif. See `scan_sequences()` for how to use this argument. Can be a single value to be recycled for all motifs, or a vector of equal length to the number of motifs.

- **threshold.type**
  - character(1) How the threshold value should be used to obtain the final threshold value in the written motif. See `scan_sequences()` for how to use this.

**Value**

NULL, invisibly.

**Author(s)**

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

**References**


**See Also**

- `read_homer()`

Other write_motifs: `write_jaspar()`, `write_matrix()`, `write_meme()`, `write_motifs()`, `write_transfac()`

**Examples**

```r
motif <- create_motif()
write_homer(motif, tempfile())
```
write_jaspar

Export motifs in JASPAR format.

Description
Convert motifs to JASPAR format and write to file. See http://jaspar.genereg.net/.

Usage
write_jaspar(motifs, file, overwrite = FALSE, append = FALSE)

Arguments
- motifs: See convert_motifs() for acceptable formats.
- file: character(1) File name.
- overwrite: logical(1) Overwrite existing file.
- append: logical(1) Add to an existing file.

Value
NULL, invisibly.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References

See Also
read_jaspar()

Other write_motifs: write_homer(), write_matrix(), write_meme(), write_motifs(), write_transfac()

Examples
transfac <- read_transfac(system.file("extdata", "transfac.txt",
                                      package = "universalmotif"))
write_jaspar(transfac, tempfile())
Export motifs as raw matrices.

Description
Write motifs as simple matrices with optional headers to file.

Usage
write_matrix(motifs, file, positions = "columns", rownames = FALSE, type,
sep = "", headers = TRUE, overwrite = FALSE, append = FALSE,
digits = 6)

Arguments
- motifs: See convert_motifs() for acceptable formats.
- file: character(1) File name.
- positions: character(1) One of c('columns', 'rows'). Partial matching allowed.
- rownames: logical(1) Include alphabet letters as rownames.
- type: character(1) One of c('PCM', 'PPM', 'PWM', 'ICM'). If missing will use whatever type the motif is currently stored as.
- sep: character(1) Indicates how to separate individual motifs. Set as NULL to have no separating lines between motifs (the default is to use a blank line).
- headers: logical(1), character(1) Indicating if and how to write names.
- overwrite: logical(1) Overwrite existing file.
- append: logical(1) Add to an existing file.
- digits: numeric(1) Number of digits to use for motif positions.

Value
NULL, invisibly.

Author(s)
Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also
read_matrix()

Other write_motifs: write_homer(), write_jaspar(), write_meme(), write_motifs(), write_transfac()

Examples
```
motif <- create_motif()
write_matrix(motif, tempfile(), headers = ">")
```
write_meme

Export motifs in MEME format.

Description

Convert motifs to minimal MEME format and write to file. See http://meme-suite.org/doc/meme-format.html.

Usage

write_meme(motifs, file, version = 5, bkg, strand, overwrite = FALSE,append = FALSE)

Arguments

motifs character(1) File name.
version numeric(1) MEME version.
bkg numeric Background letter frequencies. If missing, will use background frequencies from motif objects (if they are identical); else background frequencies will be set to freq = 1/length(alphabet)
strand character If missing, will use strand from motif objects (if identical); otherwise will default to "+-"
overwrite logical(1) Overwrite existing file.
append logical(1) Add to an existing file. Motifs will be written in minimal format, so it is recommended to only use this if the existing file is also a minimal MEME format file.

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

read_meme()
Other write_motifs: write_homer(), write_jaspar(), write_matrix(), write_motifs(), write_transfac()
write_motifs

Examples

transfac <- read_transfac(system.file("extdata", "transfac.txt",
package = "universalmotif"))
write_meme(transfac, tempfile())

write_motifs

Export motifs in universalmotif format.

Description

Write motifs as universalmotif objects to file. For optimal storage of universalmotif class motifs, consider using saveRDS() and readRDS(). Currently the universalmotif format is YAML-based, but this is subject to change.

Usage

write_motifs(motifs, file, minimal = FALSE, multifreq = TRUE,
progress = FALSE, overwrite = FALSE, append = FALSE, BP = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>motifs</td>
<td>See convert_motifs() for acceptable formats.</td>
</tr>
<tr>
<td>file</td>
<td>character(1) File name.</td>
</tr>
<tr>
<td>minimal</td>
<td>logical(1) Only write essential motif information.</td>
</tr>
<tr>
<td>multifreq</td>
<td>logical(1) Write multifreq slot, if present.</td>
</tr>
<tr>
<td>progress</td>
<td>logical(1) Show progress.</td>
</tr>
<tr>
<td>overwrite</td>
<td>logical(1) Overwrite existing file.</td>
</tr>
<tr>
<td>append</td>
<td>logical(1) Add to an existing motif file. Package version in existing motif file must be greater than 1.2.0.</td>
</tr>
<tr>
<td>BP</td>
<td>logical(1) Allows for the use of BiocParallel within write_motifs(). See BiocParallel::register() to change the default backend.</td>
</tr>
</tbody>
</table>

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

See Also

Other write_motifs: write_homer(), write_jaspar(), write_matrix(), write_meme(), write_transfac()
write_transfac  

Export motifs in TRANSFAC format.

Description

Convert motifs to TRANSFAC format and write to file.

Usage

write_transfac(motifs, file, overwrite = FALSE, append = FALSE, name.tag = "ID", altname.tag = "NA")

Arguments

motifs  See convert_motifs() for acceptable formats.
file  character(1) File name.
overwrite  logical(1) Overwrite existing file.
append  logical(1) Add to an existing file.
name.tag  character(1) The tag to use when writing the motifs name slot.
altnname.tag  character(1) The tag to use when writing the motifs altname slot. Note that no tag will be written if the slot is empty.

Details

If the family slot of a motif is not empty, then its contents will included using the HC tag. Similarly for the organism slot using the tag OS. The default name and alternate name tags are ID and NA, respectively, though these can be set manually.

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <benjamin.tremblay@uwaterloo.ca>

References


See Also

read_transfac()

Other write_motifs: write_homer(), write_jaspar(), write_matrix(), write_meme(), write_motifs()
Examples

jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
                          package = "universalmotif"))

write_transfac(jaspar, tempfile())
Index

* datasets
  ArabidopsisMotif, 5
  ArabidopsisPromoters, 6
  examplemotif, 25
  examplemotif2, 26
  fontDFroboto, 27
  JASPAR2018_CORE_DBSCORES, 30
* internal
  reexports, 54
* read_motifs
  read_cisbp, 45
  read_homer, 47
  read_jaspar, 48
  read_matrix, 49
  read_meme, 50
  read_motifs, 51
  read_transfac, 52
  read_uniprobe, 53
* write_motifs
  write_homer, 91
  write_jaspar, 93
  write_matrix, 94
  write_meme, 95
  write_motifs, 96
  write_transfac, 97
  [, universalmotif-method
    (universalmotif-class), 71
  ]<-, universalmotif-method
    (universalmotif-class), 71
  AAStringSet, 18
  add_gap (utils-motif), 76
  add_multifreq, 3
  add_multifreq(), 4, 18, 19, 23, 25, 57, 58, 61, 72, 90
  ArabidopsisMotif, 5
  ArabidopsisPromoters, 5, 6
  as.data.frame, 54
  as.data.frame (reexports), 54
  as.data.frame, universalmotif-method
    (universalmotif-class), 71
  average_ic (utils-motif), 76
  average_ic(), 7, 31, 33, 35, 44, 89
  BiocParallel::register(), 37, 51, 96
  Biostrings::injectHardMask(), 60
  Biostrings::mask(), 60
  Biostrings::matchPWM(), 60, 61
  Biostrings::oligonucleotideFrequency(), 28
  BStringSet, 18
  calc_complexity (utils-sequence), 83
  calc_complexity(), 63, 83, 84
  calc_windows (utils-sequence), 83
  calc_windows(), 84
  cbind, 54
  cbind (reexports), 54
  cbind, universalmotif-method
    (universalmotif-class), 71
  colMeans, 54
  colMeans (reexports), 54
  colMeans, universalmotif-method
    (universalmotif-class), 71
  colnames, 54
  colnames (reexports), 54
  colnames, universalmotif-method
    (universalmotif-class), 71
  colSums, 54
  colSums (reexports), 54
  colSums, universalmotif-method
    (universalmotif-class), 71
  compare_columns (utils-motif), 76
  compare_motifs, 6
  compare_motifs(), 7, 30–32, 34–36, 43–45, 77, 90
  consensus_to_ppm (utils-motif), 76
  consensus_to_ppm(), 79
  consensus_to_ppmAA (utils-motif), 76
consensus_to_ppmAA(), 79
convert_motifs, 10
convert_motifs(), 4–6, 9, 12, 14–16, 23, 26, 27, 33–36, 38, 42, 43, 57, 65, 68, 70, 88, 92–97
convert_motifs, AsIs-method
(convert_motifs), 10
convert_motifs, ICMatrix-method
(convert_motifs), 10
convert_motifs, list-method
(convert_motifs), 10
convert_motifs, matrix-method
(convert_motifs), 10
convert_motifs, Motif-method
(convert_motifs), 10
convert_motifs, MotifList-method
(convert_motifs), 10
convert_motifs, PCM-method
(convert_motifs), 10
convert_motifs, pwm-method
(convert_motifs), 10
convert_motifs, PFMatrix-method
(convert_motifs), 10
convert_motifs, PWM-method
(convert_motifs), 10
convert_motifs, pfm-method
(convert_motifs), 10
convert_motifs, PFMMatrix-method
(convert_motifs), 10
convert_motifs, TFFMFirst-method
(convert_motifs), 10
convert_motifs, universalmotif-method
(convert_motifs), 10
convert_motifs, XMatrixList-method
(convert_motifs), 10
convert_type, 14
convert_type(), 7, 19, 33, 35, 39, 44, 70, 89
count_klets (utils-sequence), 83
count_klets(), 63, 84
create_motif, 16
create_motif(), 22, 31, 57, 68, 70, 71, 79
count_klets (utils-sequence), 83
count_klets(), 63, 84
create_motif, AAStringSet-method
(create_motif), 16
create_motif, BStringSet-method
(create_motif), 16
create_motif, character-method
(create_motif), 16
create_motif, DNAStringSet-method
(create_motif), 16
create_motif, matrix-method
(create_motif), 16
create_motif, missing-method
(create_motif), 16
create_motif, numeric-method
(create_motif), 16
create_motif, RNAStringSet-method
(create_motif), 16
count_klets, 83
count_klets(), 63, 84
create_motif, AsIs-method
(create_motif), 10
create_motif, ICMatrix-method
(create_motif), 10
convert_motifs, list-method
(convert_motifs), 10
convert_motifs, matrix-method
(convert_motifs), 10
convert_motifs, Motif-method
(convert_motifs), 10
convert_motifs, MotifList-method
(convert_motifs), 10
convert_motifs, PCM-method
(convert_motifs), 10
convert_motifs,_pwm-method
(convert_motifs), 10
create_sequences, 21
create_sequences(), 19, 21, 22, 29, 57, 67, 83
create_motif, AAStringSet-method
(create_motif), 16
create_motif, BStringSet-method
(create_motif), 16
create_motif, character-method
(create_motif), 16
create_motif, DNAStringSet-method
(create_motif), 16
create_motif, matrix-method
(create_motif), 16
create_motif, missing-method
(create_motif), 16
create_motif, numeric-method
(create_motif), 16
create_motif, RNAStringSet-method
(create_motif), 16
dataFrame, 28
dnastring, 60
dnastringset, 6, 18, 60
enrich_motifs, 22
enrich_motifs(), 61, 67
example_motif, 25
example_motif2, 26
get_klets, 83
get_klets(), 84
ggplot2::ggplot(), 45
ggtree::ggtree(), 43–45
get_consensus (utils-motif), 76
get_consensus(), 79
get_consensusAA (utils-motif), 76
get_consensusAA(), 79
getScores (utils-motif), 76
getScores(), 41
JASPAR2018_CORE_DBSCORES, 30
log_string_pval (utils-motif), 76
log_string_pval(), 50, 79
make_DBscores, 30
make_DBscores(), 8, 9, 30, 32
mask_ranges (utils-sequence), 83
mask_ranges(), 63, 84
mask_seqs (utils-sequence), 83
mask_seqs(), 60, 63, 84
MaskedXString, 60
meme_alph (utils-sequence), 83
meme_alph(), 55, 84
merge_motifs, 32
merge_motifs(), 34, 36
merge_similar, 34
merge_similar(), 34
motif_peaks, 36
motif_peaks(), 37
motif_pvalue, 38
motif_pvalue(), 23–25, 39, 40, 58–61
motif_range (utils-motif), 76
motif_range(), 41, 79
motif_rc, 42
motif_rc(), 42
motif_score (utils-motif), 76
motif_score(), 41, 79
motif_tree, 43
motif_tree(), 9, 43
motifStack::motifStack(), 45
ncol, 54
ncol (reexports), 54
ncol, universalmotif-method
  (universalmotif-class), 71
normalize, 54
normalize (reexports), 54
normalize, universalmotif-method
  (universalmotif-class), 71
nrow, 54
nrow (reexports), 54
nrow, universalmotif-method
  (universalmotif-class), 71
pcm_to_ppm (utils-motif), 76
pcm_to_ppm(), 79
position_icscore (utils-motif), 76
position_icscore(), 79
ppm_to_icm (utils-motif), 76
ppm_to_icm(), 79
ppm_to_pcm (utils-motif), 76
ppm_to_pcm(), 79
ppm_to_pwm (utils-motif), 76
prob_match (utils-motif), 76
prob_match(), 41, 79
prob_match_bkg (utils-motif), 76
prob_match_bkg(), 41
processx::run(), 56, 57
pwm_to_ppm (utils-motif), 76
pwm_to_ppm(), 79
read_cisbp, 45, 47–49, 51–54
read_homer, 46, 47, 48, 49, 51–54
read_homer(), 92
read_jaspar, 46, 47, 48, 49, 51–54
read_jaspar(), 93
read_matrix, 46–48, 49, 51–54
read_matrix(), 94
read_meme, 46–49, 50, 52–54
read_meme(), 56, 57, 95
read_motifs, 46–49, 51, 53, 54
read_motifs(), 51
readtransf, 46–49, 51, 52, 54
readtransf(), 97
readuniprobe, 46–49, 51–53, 53
readRDS(), 51, 96
reexports, 54
requires_update (tidy-motifs), 68
requires_update(), 69
RNAStringSet, 18, 60
round_motif (utils-motif), 76
round_motif(), 79
rowMeans, 54
rowMeans (reexports), 54
rowMeans, universalmotif-method
  (universalmotif-class), 71
rownames, 54
rownames (reexports), 54
rownames, universalmotif-method
  (universalmotif-class), 71
rowSums, 54
rowSums (reexports), 54
rowSums, universalmotif-method
  (universalmotif-class), 71
run_meme, 54
run_meme(), 54, 56
sample(), 22, 39, 66
sample_sites, 57
saveRDS(), 51, 96
scan_sequences, 58
scan_sequences(), 4, 5, 23–25, 29, 36, 38, 47, 58, 60, 67, 92
score_match (utils-motif), 76
score_match(), 41, 79
sequence_complexity, 61
sequence_complexity(), 62, 83, 85
show, universalmotif-method
  (universalmotif-class), 71
shuffle_motifs, 65
shuffle_motifs(), 19, 31, 65, 67
shuffle_sequences, 66
shuffle_sequences(), 22, 23, 25, 29, 57, 65, 66, 84, 85
shuffle_string (utils-sequence), 83
shuffle_string(), 84, 85
slide_fun (utils-sequence), 83
slide_fun(), 85
stats::fisher.test(), 24
stats::p.adjust(), 23
subset, 54
subset (reexports), 54
subset, universalmotif-method
  (universalmotif-class), 71
summarise_motifs (utils-motif), 76
summarise_motifs(), 79
switch_alph, 68
tidy-motifs, 68
to_df (tidy-motifs), 68
to_df(), 12, 69
to_list (tidy-motifs), 68
to_list(), 12, 69
trim_motifs, 70
ungap (utils-motif), 76
universalmotif, 4, 5, 12, 14, 18, 25, 26, 43, 46–50, 52, 53, 72, 73, 77, 79
universalmotif (universalmotif-class), 71
universalmotif-class, 71
universalmotif-pkg, 75
update_motifs (tidy-motifs), 68
update_motifs(), 69
utilities, 75
utils-motif, 75, 76
utils-sequence, 75, 82
view_logo, 86
view_logo(), 90
view_motifs, 88
view_motifs(), 9, 27, 86, 87, 89, 90
window_string (utils-sequence), 83
window_string(), 85
write_homer, 91, 93–97
write_jaspar, 92, 93, 94–97
write_matrix, 92, 93, 94, 95–97
write_meme, 92–94, 95, 96, 97
write_motifs, 92–95, 96, 97
write_motifs(), 4, 5, 51, 96
write_transfac, 92–96, 97
XStringSet, 4, 17, 21–23, 28, 55, 57, 58, 60, 61, 66, 67