Package 'universalmotif'

April 16, 2019

Title Import, Modify, and Export Motifs with R

Version 1.0.22

URL https://github.com/bjmt/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, ggplot2, ape, ggtree, ggseqlogo, stats, utils, gtools, Rdpack (>= 0.7), Biostrings, BiocGenerics, Rcpp, processx

Suggests knitr, BiocStyle, TFBSTools, MotIV, PWMEnrich, motifStack, seqLogo, rmarkdown, MotifDb, testthat, rGADEM, motifRG, motifbreakR, RcisTarget, motifcounter, Logolas, BiocParallel, BCRANK

LinkingTo Rcpp

RdMacros Rdpack

VignetteBuilder knitr

biocViews MotifAnnotation, MotifDiscovery, DataImport, GeneRegulation

RoxygenNote 6.1.1

Roxygen list(markdown = TRUE)

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' 'merge_motifs.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'

'shuffle_motifs.R' 'shuffle_sequences.R' 'switch_alph.R'
'trim_motifs.R' 'universalmotif-methods.R' 'universalmotif.R'
'utils.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_8

git_last_commit 34a663f

git_last_commit_date 2019-03-06

Date/Publication 2019-04-15

Author Benjamin Jean-Marie Tremblay [aut, cre]

Maintainer Benjamin Jean-Marie Tremblay <b2tremblay@uwaterloo.ca>

R topics documented:

`add_multifreq` *Add multi-letter information to a motif.*

Description

If the original sequences are available for a particular motif, then they can be used to generate higher-order PPM matrices.

Usage

```
add_multifreq(motif, sequences, add.k = 2:3, RC = FALSE,
  threshold = 0.01, threshold.type = "logodds", motifs.perseq = 1)
```

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 right sequence.
`add.k`	`numeric(1)` The k-let lengths to add.
`RC`	`logical(1)` Check the reverse complement of a DNA sequence. See `scan_sequences()`.
`threshold`	`numeric(1)` Between 0 and 1. See `scan_sequences()`.
`threshold.type`	`character(1)` One of `c('logodds', 'pvalue')`. See `scan_sequences()`.
`motifs.perseq`	`numeric(1)` If `scan_sequences()` is run, then this indicates how many hits from each sequence is to be used.

Details

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** enrivonment. Despite this, if you wish it can be preserved in text using write_motifs().

Note: 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 universalmotif object with filled `multifreq` slot.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@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* `universalmotif` *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`

Usage

```
ArabidopsisMotif
```

Format

universalmotif

ArabidopsisPromoters *Arabidopsis promoters as a* DNAStringSet.

Description

50 Arabidopsis promoters, each 1000 bases long. See the 'Advanced usage' vignette, section 7, for an example workflow describing extracting promoter sequences.

Usage

```
ArabidopsisPromoters
```

Format

DNAStringSet

compare_motifs *Compare motifs.*

Description

Compare motifs using four available metrics: Pearson correlation coefficient (Pietrokovski 1996), Euclidean distance (Choi et al. 2004), Sandelin-Wasserman similarity (Sandelin and Wasserman 2004), and Kullback-Leibler divergence (Roepcke et al. 2005).

Usage

```
compare_motifs(motifs, compare.to, db.scores, use.freq = 1,
  use.type = "PPM", method = "MPCC", tryRC = TRUE, min.overlap = 6,
  min.mean.ic = 0.5, relative_entropy = FALSE,
  normalise.scores = FALSE, max.p = 0.01, max.e = 10,
  progress = TRUE, BP = FALSE)
```

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 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.
method	character(1) One of c('PCC', 'MPCC', 'EUCL', 'MEUCL', 'SW', 'MSW', 'KL', 'MKL'). See details.
tryRC	logical 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 less than 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.

`relative_entropy` `logical(1)` For ICM calculation. 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.

`progress` `logical(1)` Show progress. Not recommended if `BP = TRUE`.

`BP` `logical(1)` Allows the use of **BiocParallel** within `compare_motifs()`. See `BiocParallel::register()` to change the default backend. Setting `BP = TRUE` is only recommended for comparing large numbers of motifs (>10,000). Furthermore, the behaviour of `progress = TRUE` is changed if `BP = TRUE`; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Details

Comparisons are calculated between two motifs at a time. All possible alignments are scored, and the best score is reported. Scores are calculated per position and summed, unless the 'mean' version of the specific metric is chosen. If using a similarity metric, then the sum of scores will favour comparisons between longer motifs; and for distance metrics, the sum of scores will favour comparisons between short motifs. This can be avoided by using the 'mean' of scores.

- PCC: Pearson correlation coefficient
  Per position:
  `PCC = sum(col1 * col2) / sqrt(sum(col1^2) * sum(col2^2))`
- MPCC: Mean PCC
  `MPCC = mean(PCC)`
- EUCL: Euclidian distance
  Per position:
  `EUCL = sqrt(sum((col1 - col2)^2)) / sqrt(2)`
- MEUCL: Mean EUCL
  `MEUCL = sum(EUCL) / ncol(alignment)`
- SW: Sandelin-Wasserman similarity
  Per position:
  `SW = 2 - sum((col1 - col2)^2)`

- MSW: Mean SW
  `MSW = mean(SW)`
- KL: Kullback-Leibler divergence
  Per position:
  `KL = 0.5 * sum(col1 * log(col1/col2) + col2 * log(col2/col1))`
- MKL: Mean Kullback-Leibler divergence
  `MKL = mean(KL)`

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; `data.frame` otherwise.

- PCC: 0 represents complete distance, >0 similarity.
- MPCC: 0 represents complete distance, 1 complete similarity.
- EUCL: 0 represents complete similarity, >0 distance.
- MEUCL: 0 represents complete similarity, sqrt(2) complete distance.
- SW: 0 represents complete distance, >0 similarity.
- MSW: 0 represents complete distance, 2 complete similarity.
- KL: 0 represents complete similarity, >0 distance.
- MKL: 0 represents complete similarity, >0 complete distance.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Choi I, Kwon J, Kim S (2004). "Local feature frequency profile: a method to measure structural similarity in proteins." *PNAS*, **101**, 3797–3802.

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon J, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas D, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman W, Parcy F, Mathelier A (2018). "JASPAR 2018: update of the open-access database of transcription factor binding profiles and its web framework." *Nucleic Acids Research*, **46**, D260-D266.

Pietrokovski S (1996). "Searching databases of conserved sequence regions by aligning protein multiple-alignments." *Nucleic Acids Research*, **24**, 3836–3845.

Roepcke S, Grossmann S, Rahmann S, Vingron M (2005). "T-Reg Comparator: an analysis tool for the comparison of position weight matrices." *Nucleic Acids Research*, **33**, W438–W441.

Sandelin A, Wasserman W (2004). "Constrained binding site diversity within families of transcription factors enhances pattern discovery bioinformatics." *Journal of Molecular Biology*, **338**(2), 207–215.

See Also

convert_motifs(), motif_tree(), view_motifs()

Examples

```
motif1 <- create_motif()
motif2 <- create_motif()
motif1vs2 <- compare_motifs(list(motif1, motif2), method = "MPCC")
## to get a dist object:
as.dist(1 - motif1vs2)
```

---

convert_motifs *Convert motif class.*

Description

Allows for easy transfer of motif information between different classes 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 needed to convert beforehand.

Usage

```
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 'PWMatrix'
convert_motifs(motifs,
  class = "universalmotif-universalmotif")

## S4 method for signature 'ICMatrix'
convert_motifs(motifs,
```

```
  class = "universalmotif-universalmotif")

## 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

The following packge-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
- matrix

The following package-class combinations can be output:

- MotIV-pwm2
- TFBSTools-PFMatrix
- TFBSTools-PWMatrix
- TFBSTools-ICMatrix
- TFBSTools-TFFMFirst
- seqLogo-pwm
- motifStack-pcm
- motifStack-pfm
- PWMEnrich-PWM
- Biostrings-PWM (`type = 'log2prob'`)
- rGADEM-motif
- universalmotif-universalmotif

Value

Single motif object or list.

Methods (by class)

- `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**)
- `PWMatrix`: Convert PWMatrix 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.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Bembom O (2018). *seqLogo: Sequence logos for DNA sequence alignments*. R package version 1.46.0.

Droit A, Gottardo R, Robertson G, Li L (2014). *rGADEM: de novo motif discovery*. R package version 2.28.0.

Mercier E, Gottardo R (2014). *MotIV: Motif Identification and Validation*. R package version 1.36.0.

Ou J, Wolfe SA, Brodsky MH, Zhu LJ (2018). "motifStack for the analysis of transcription factor binding site evolution." *Nature Methods*, **15**, 8-9. doi: 10.1038/nmeth.4555, http://dx.doi.org/10.1038/nmeth.4555.

Pagès H, Aboyoun P, Gentleman R, DebRoy S (2018). *Biostrings: Efficient manipulation of biological strings*. R package version 2.48.0.

Shannon P, Richards M (2018). *MotifDb: An Annotated Collection of Protein-DNA Binding Sequence Motifs*. R package version 1.22.0.

Stojnic R, Diez D (2015). *PWMEnrich: PWM enrichment analysis*. R package version 4.16.0.

Tan G, Lenhard B (2016). "TFBSTools: an R/Bioconductor package for transcription factor binding site analysis." *Bioinformatics*, **32**, 1555-1556. doi: 10.1093/bioinformatics/btw024, http://bioinformatics.oxfordjournals.org/content/32/10/1555.

Yao Z (2012). *motifRG: A package for discriminative motif discovery, designed for high throughput sequencing dataset*. R package version 1.24.0.

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
}
```

convert_type *Convert universalmotif type.*

Description

Switch between position count matrix (PCM), position probability matrix (PPM), position weight matrix (PWM), and information count matrix (ICM) types.

Usage

```
convert_type(motifs, type, pseudocount, nsize_correction = FALSE,
  relative_entropy = FALSE)
```

Arguments

motifs	See convert_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 apearing in PWM matrices. If missing, the pseudocount stored in the universalmotif 'pseudocount' slot will be used, which defaults to 0.8; the suggested value from Nishida et al. (2009).
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

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.

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)).

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 (log2(letter_probability / 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-likelihodd of these letters will be -Inf.

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: totalIC - Hi, where the total information totalIC

`totalIC <- log2(alphabet_length)`, and the Shannon entropy (Shannon 1948) for a specific position (Hi)

`Hi <- -sum(sapply(alphabet_frequencies, 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 `nsize_correction = TRUE`, however it will only be applied if the 'nsites' slot is not empty. This is done using `TFBSTools:::schneider_correction` (Tan and Lenhard 2016). 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 `letter_freq * log2(letter_freq / bkg_freq)`. When calculating information content using Shannon entropy, the maximum content for each position will always be `log2(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 `convert_motifs()` for possible output motif objects.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Kullback S, Leibler RA (1951). "On information and sufficiency." *The Annals of Mathematical Statistics*, **22**, 79-86.

Nishida K, Frith MC, Nakai K (2009). "Pseudocounts for transcription factor binding sites." *Nucleic Acids Research*, **37**, 939-944.

Schneider TD, Stormo GD, Gold L, Ehrenfeucht A (1986). "Information content of binding sites on nucleotide sequences." *Journal of Molecular Biology*, **188**, 415-431.

Schneider TD, Stephens RM (1990). "Sequence Logos: A New Way to Display Consensus Sequences." *Nucleic Acids Research*, **18**, 6097-6100.

Schneider TD (1991). "Theory of Molecular Machines. II. Energy Dissipation from Molecular Machines." *Journal of Theoretical Biology*, **148**, 125-137.

Shannon CE (1948). "A Mathematical Theory of Communication." *Bell System Technical Journal*, **27**, 379-423.

Stormo GD, Schneider TD, Gold L, Ehrenfeucht A (1982). "Use of the Perceptron algorithm to distinguish translational initiation sites in E. coli." *Nucleic Acids Research*, **10**, 2997-3011.

Tan G, Lenhard B (2016). "TFBSTools: an R/Bioconductor package for transcription factor binding site analysis." *Bioinformatics*, **32**, 1555-1556. doi: 10.1093/bioinformatics/btw024, http://bioinformatics.oxfordjournals.org/content/32/10/1555.

See Also

convert_motifs()

Examples

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

## pseudocounts default to 0.8
jaspar.pwm <- convert_type(jaspar.pcm, type = "PPM")

## setting pseudocounts to 0 will prevent any correction from being
## applied to PPM/PWM matrices
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.

Usage

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

Arguments

input	character, numberic, matrix, XStringSet, or missing
alphabet	character(1) One of c('DNA', 'RNA', 'AA', 'custom'), or a combined string representing the letters.
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 apearing in PWM matrices.
bkg	numeric Must sum to 1 and be equal in length to the alphabet length.
nsites	numeric(1) Number of sites the motif was constructed from.
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.
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(1) If the motif is created from a set of sequences, then the add_multifreq() function can be run at the same type.

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. Furthermore, be careful about the `nsites` slot when creating motifs from consensus strings: for example, the following call `create_motif("TAAAT")` generates a motif with `nsites = 1`.

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`.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

See Also

`convert_type()`, `add_multifreq()`, `create_sequences()`, `shuffle_motifs()`.

`create_sequences()`

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
DNA.motif <- create_motif("TTTTTTT", nsites = 10)

# if ambiguity letters are found and nsites is not specified, nsites will
# be set to the minimum required to respect amibiguity letters
DNA.motif <- create_motif("TATAWAW")
DNA.motif <- create_motif("NNVVWWAAWWDDN")

# be careful about setting nsites when using ambiguity letters!
DNA.motif <- create_motif("NNVVWWAAWWDDN", 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 = "custom")
# 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 = "custom")

# 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")

## End(Not run)

##### create motifs from XStringSet objects

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)

##### create motifs with filled 'multifreq' slot

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)

# 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

DNA.motif <- 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,
  monofreqs, difreqs, trifreqs, progress = FALSE, BP = FALSE)
```

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.
monofreqs	numeric Alphabet frequencies to use. If missing assumes uniform frequencies. Not used if difreq or trifreq are input.
difreqs	numeric Dinucleotide frequencies. DNA/RNA only. Must be a named numeric vector of length 16.
trifreqs	numeric Trinucleotide frequencies. DNA/RNA only. Must be a named numeric vector of length 64.
progress	logical(1) Show progress. Not recommended if BP = TRUE.
BP	logical(1) Allows the use of **BiocParallel** within create_sequences(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for large jobs (such as create_sequences(seqlen=100000,seqnum=100000)) Furthermore, the behaviour of progress = TRUE is changed if BP = TRUE; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Value

XStringSet The returned sequences are *unnamed*.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Pagès H, Aboyoun P, Gentleman R, DebRoy S (2018). *Biostrings: Efficient manipulation of biological strings*. R package version 2.48.0.

See Also

create_motif(), shuffle_sequences()

Examples

```
## create DNA sequences with slightly increased AT content:
sequences <- create_sequences(monofreqs = c(0.3, 0.2, 0.2, 0.3))
## create custom sequences:
sequences.QWER <- create_sequences("QWER")
## you can include non-alphabet characters are well, even spaces:
sequences.custom <- create_sequences("!@#$ ")
```

---

enrich_motifs *Enrich for input motifs in a set of sequences.*

Description

Given a set of target and background sequences, test if the input motifs are significantly enriched in the targets sequences relative to the background sequences.

Usage

```
enrich_motifs(motifs, sequences, bkg.sequences, search.mode = "hits",
  max.p = 1e-05, max.q = 1e-05, max.e = 0.001, qval.method = "fdr",
  positional.test = "t.test", threshold = 0.001,
  threshold.type = "pvalue", verbose = 1, RC = FALSE, use.freq = 1,
  shuffle.k = 2, shuffle.method = "linear",
  shuffle.leftovers = "asis", return.scan.results = FALSE,
  progress = TRUE, BP = FALSE)
```

Arguments

motifs	See convert_motifs() for acceptable motif formats.
sequences	XStringSet Alphabet should match motif.
bkg.sequences	XStringSet Optional; if missing, shuffle_sequences() is used to create background sequences from the input sequences.
search.mode	character(1) One of c('hits', 'positional', 'both'). See details.
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 adjusted P-value with the number of input motifs times two (McLeay and Bailey 2010).
qval.method	character(1) See stats::p.adjust().
positional.test	character(1) One of c('t.test', 'wilcox.test', 'chisq.test', 'shapiro.test'). If using the Shapiro test for normality, then only the input sequences are tested for positionality; the background sequences are ignored. See stats::t.test(), stats::wilcox.test(), stats::chisq.test(), stats::shapiro.test().
threshold	numeric(1) Between 1 and 0. See scan_sequences().
threshold.type	character(1) One of c('logodds', 'pvalue'). See scan_sequences().
verbose	numeric(1) 0 for no output, 4 for max verbosity.
RC	logical(1) Whether to consider the reverse complement of the sequences. Only available for DNAStringSet, RNAStringSet sequences.
use.freq	numeric(1) If the multifreq slot of the motifs are filled, then they can be used to scan the sequences. See scan_sequences().
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('markov', 'linear', 'random'). See shuffle_sequences().
shuffle.leftovers	character(1) One of c('asis', 'first', 'split', 'discard'). Only used if shuffle.method = 'random'. See shuffle_sequences().
return.scan.results	logical(1) Return output from scan_sequences().
progress	logical(1) Show progress. Note recommended if BP = TRUE. Set to FALSE if verbose = 0

BP — logical(1) Allows the use of **BiocParallel** within enrich_motifs(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for exceptionally large jobs (be wary of memory usage however, as enrich_motifs() does not try and limit itself in this regard). Furthermore, the behaviour of progress = TRUE is changed if BP = TRUE; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Details

To find enriched motifs, scan_sequences() is run on both target and background sequences. If search.mode = 'hits', stats::fisher.test() is run to test for enrichment. If search.mode = 'positional', then the test as set by positional.test is run to check for positional differences between target and background sequences. However if positional.test = 'shapiro.test', then only target sequence hits are considered.

Value

data.frame Motif enrichment results. The resulting data.frame contains the following columns: * motif Motif name. * total.seq.hits Total number of matches accross all target sequences. * num.seqs.hits Number of target sequences which contain matches. * num.seqs.total Number of target sequences. * total.bkg.hits Total number of matches accross all background sequences. * num.bkg.hits Number of background sequences which contain matches. * num.bkg.total Number of background sequences. * Pval.hits P-value of enrichment. Only shown if search.mode = c('hits', 'bot
* Qval.hits Q-val of enrichment. Only shown if search.mode = c('hits', 'both'). * Eval.hits E-val of enrichment. Only shown if search.mode = c('hits', 'both'). * Pval.pos P-value of positional comparison. Only shown if search.mode = c('positional', 'both'). * Qval.pos Q-value of positional comparison. Only shown if search.mode = c('positional', 'both'). * Eval.pos E-value of positional comparison. Only shown if search.mode = c('positional', 'both').

Author(s)

Benjamin Jean-Marie Tremblay <b2tremblay@uwaterloo.ca>

References

McLeay R, Bailey T (2010). "Motif Enrichment Analysis: A unified framework and method evaluation." *BMC Bioinformatics*, **11**.

See Also

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

Examples

```
data(ArabidopsisPromoters)
data(ArabidopsisMotif)
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

examplemotif2 *Another example motif in* universalmotif *format.*

Description

A simple DNA motif with a non-empty `multifreq` slot. See the 'Advanced usage' vignette, section 2, for manually recreating this motif.

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 they will be converted for the duration of the `filter_motifs()` operation.

Usage

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

Arguments

motifs	list See convert_motifs() for acceptable formats.
name	character Keep motifs by names.
altname	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.

Value

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

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

Examples

```
## By minimum IC:
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
                                  package = "universalmotif"))
jaspar.ic10 <- filter_motifs(jaspar, icscore = 10)

## By organism:
if (requireNamespace("MotifDb", quietly = TRUE) && .Platform$OS.type == "unix") {
  library(MotifDb)
  motifs <- convert_motifs(MotifDb)
  motifs <- filter_motifs(motifs, organism = c("Athaliana", "Mmusculus"))
}
```

JASPAR2018_CORE_DBSCORES

JASPAR2018 CORE database scores.

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, with normalise.scores = FALSE. This particular set of scores is organized as a list, with each list item being a data.frame of scores for a specific comparison method. In each data.frame is the mean and sd of scores resulting between the comparisons of JASPAR2018 CORE motifs with randomized motifs of the specifed subject and target motif length.

Usage

```
JASPAR2018_CORE_DBSCORES
```

Format

list with three data.frames

JASPAR2018_CORE_DBSCORES_NORM

JASPAR2018 CORE database scores (normalised).

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, with normalise.scores = TRUE. This particular set of scores is organized as a list, with each list item being a data.frame of scores for a specific comparison method. In each data.frame is the mean and sd of scores resulting between the comparisons of JASPAR2018 CORE motifs with randomized motifs of the specifed subject and target motif length.

Usage

```
JASPAR2018_CORE_DBSCORES_NORM
```

Format

list with three data.frames

make_DBscores *Utility functions.*

Description

Various small functions used for motif creation. Note that all of the functions described here (with the following exceptions: `make_DBscores()`, `summarise_motifs()`) are here only for demonstration purposes; internally the **universalmotif** package uses faster C++ code for type conversion and consensus manipulation.

Usage

```
make_DBscores(db.motifs, method, shuffle.db = TRUE, shuffle.k = 3,
  shuffle.method = "linear", shuffle.leftovers = "asis",
  rand.tries = 1000, normalise.scores = TRUE, min.overlap = 6,
  min.mean.ic = 0, progress = TRUE, BP = FALSE)

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

icm_to_ppm(position)

pcm_to_ppm(position, pseudocount = 0.8)

ppm_to_pcm(position, nsites = 100)

ppm_to_pwm(position, bkg, pseudocount = 0.8, nsites = 100,
  smooth = TRUE)

pwm_to_ppm(position, bkg)

position_icscore(position, bkg, type, pseudocount = 0.8, nsites = 100,
  relative_entropy = FALSE)

get_consensus(position, alphabet = "DNA", type = "PPM",
  pseudocount = 0.8)

consensus_to_ppm(letter)

consensus_to_ppmAA(letter)

get_consensusAA(position, type, pseudocount)

summarise_motifs(motifs, na.rm = TRUE)
```

Arguments

db.motifs	`list` Database motifs.
method	`character(1)` One of `c('PCC', 'MPCC', 'EUCL', 'MEUCL', 'SW', 'MSW', 'KL', 'MKL')`. See `compare_motifs()`.

shuffle.db logical(1) Shuffle db.motifs rather than generate random motifs with create_motif().

shuffle.k numeric(1) See shuffle_motifs().

shuffle.method character(1) See shuffle_motifs().

shuffle.leftovers
character(1) See shuffle_motifs().

rand.tries numeric(1) Number of random motifs to create for P-value computation.

normalise.scores
logical(1) See compare_motifs().

min.overlap numeric(1) Minimum required motif overlap. See compare_motifs().

min.mean.ic numeric(1) See compare_motifs().

progress logical(1) Show progress. Not recommended if BP = TRUE.

BP logical(1) Use the **BiocParallel** package. See BiocParallel::register() to change the default backend.

position numeric A numeric vector representing the frequency or probability for each alphabet letter at a specific position.

bkg Numeric Should be the same length as the alphabet length.

schneider_correction
logical(1) Apply sample size correction.

nsites numeric(1) Number of sites motif originated from.

relative_entropy
logical(1) Calculate information content as relative entropy or Kullback-Leibler divergence.

pseudocount numeric(1) Used to prevent zeroes in motif matrix.

smooth logical(1) Apply pseudocount correction.

type character(1) One of c('PCM', 'PPM', 'PWM' 'ICM').

alphabet character(1) One of c('DNA', 'RNA').

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

motifs list A list of universalmotif motifs.

na.rm logical Remove columns where all values are NA.

Value

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 consensus_to_ppm() and consensus_to_ppmAA(): a numeric vector of length 4 and 20, respectively.

For position_icscore(): a numeric vector of length 1.

For get_consensus() and get_consensusAA(): a character vector of length 1.

For make_DBscores(): a data.frame with score distributions for the input database.

For summarise_motifs(): a data.frame with columns representing the universalmotif slots.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

See Also

create_motif(), compare_motifs()

Examples

```
## make_DBscores
## Generate P-value database for use with compare_motifs. Note that these
## must be created individually for all combinations of methods and
## normalisation.
## Not run:
library(MotifDb)
motifs <- convert_motifs(MotifDb[1:100])
make_DBscores(motifs, method = "PCC")

## End(Not run)

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

## icm_to_ppm
## Do the opposite of ppm_to_icm.
motif.ppm <- apply(motif.icm, 2, icm_to_ppm)

## pcm_to_ppm
## Go from a count type motif to a probability type motif.
motif.pcm <- create_motif(type = "PCM", nsites = 50)["motif"]
motif.ppm2 <- apply(motif.pcm, 2, pcm_to_ppm, pseudocount = 1)

## ppm_to_pcm
## Do the opposite of pcm_to_ppm.
motif.pcm2 <- apply(motif.ppm2, 2, ppm_to_pcm, nsites = 50)

## ppm_to_pwm
## Go from a probability type motif to a weight type motif.
motif.pwm <- apply(motif.ppm, 2, ppm_to_pwm, nsites = 100,
                   bkg = c(0.25, 0.25, 0.25, 0.25))

## pwm_to_ppm
## Do the opposite of ppm_to_pwm.
motif.ppm3 <- apply(motif.pwm, 2, pwm_to_ppm,
                    bkg = c(0.25, 0.25, 0.25, 0.25))

## Note that not all type conversions can be done directly; for those
## type conversions which are unavailable, universalmotif just chains
## together others (i.e. from PCM -> ICM => pcm_to_ppm -> ppm_to_icm)

## position_icscore
## Similar to ppm_to_icm, except this calculates a sum for the position.
ic.scores <- apply(motif.ppm, 2, position_icscore, type = "PPM",
                   bkg = c(0.25, 0.25, 0.25, 0.25))

## get_consensus
```

```
## Get a consensus string from a DNA/RNA motif.
motif.consensus <- apply(motif.ppm, 2, get_consensus)

## consensus_to_ppm
## Do the opposite of get_consensus. Note that loss of information is
## inevitable.
motif.ppm4 <- sapply(motif.consensus, consensus_to_ppm)

## 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.
motif.aa <- create_motif(alphabet = "AA")["motif"]
motif.aa.consensus <- apply(motif.aa, 2, get_consensusAA, type = "PPM")

## consensus_to_ppmAA
## Do the opposite of get_consensusAA.
motif.aa2 <- sapply(motif.aa.consensus, consensus_to_ppmAA)

## summarise_motifs
## 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))
```

`merge_motifs` *Merge motifs.*

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.

Usage

```
merge_motifs(motifs, method = "MPCC", use.type = "PPM",
  min.overlap = 6, min.mean.ic = 0.5, tryRC = TRUE,
  relative_entropy = FALSE, normalise.scores = FALSE)
```

Arguments

`motifs`	See `convert_motifs()` for acceptable motif formats.
`method`	`character(1)` One of `c('PCC', 'MPCC', 'EUCL', 'MEUCL', 'SW', 'MSW', 'KL', 'MKL')`. 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`.
`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 less than 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.

`tryRC` `logical` Try the reverse complement of the motifs as well, report the best score.

`relative_entropy` `logical(1)` For ICM calculation. 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.

Value

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

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

See Also

`compare_motifs()`

Examples

```
if (requireNamespace("MotifDb", quietly = TRUE) && .Platform$OS.type == "unix") {
  library(MotifDb)
  merged.motif <- merge_motifs(MotifDb[1:5])
}
```

---

motif_pvalue *Motif P-value and scoring utility*

Description

For calculating p-values/logodds scores for any number of motifs.

Usage

```
motif_pvalue(motifs, score, pvalue, bkg.probs, use.freq = 1, k = 6,
  progress = TRUE, BP = FALSE)
```

Arguments

`motifs` See `convert_motifs()` for acceptable motif formats.

`score` `numeric` Get a p-value for a motif from a logodds score.

`pvalue` `numeric` Get a logodds score for a motif from a p-value.

bkg.probs `numeric, list` If supplying individual background probabilities for each motif, a list. If missing, assumes a uniform background. Currently does not supported if `use.freq > 1`.

use.freq `numeric(1)` By default uses the regular motif matrix; otherwise uses the corresponding `multifreq` matrix.

k `numeric(1)` For speed, scores/p-values can be approximated after subsetting the motif every `k` columns. If `k` is a value equal or higher to the size of input motif(s), then the calculations are exact.

progress `logical(1)` Show progress. Not recommended if `BP = TRUE`.

BP `logical(1)` Allows the use of **BiocParallel** within `motif_pvalue()`. See `BiocParallel::register` to change the default backend. Setting `BP = TRUE` is only recommended for exceptionally large jobs. Furthermore, the behaviour of `progress = TRUE` is changed if `BP = TRUE`; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Details

Calculating p-values 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 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 based on a given p-value, the function simply guesses different scores until it finds one which when used to calculate a p-value, returns a p-value reasonably close to the given p-value. For low p-values, this usually only takes a couple of guesses.

Note that the approximation is *mostly* on the conservative side. The higher the number of motif subsets, the worse the approximation. Furthermore, the speed of `motif_pvalue()` increases with a decreasing p-value.

Value

`numeric` A vector of scores/p-values.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Hartmann H, Guthöhrlein E, Siebert M, and S. Luehr J. Söding (2013). "P-value-based regulatory motif discovery using positional weight matrices." *Genome Research*, **23**, 181–194.

Examples

```
data(examplemotif)

## p-value/score calculations are performed using the PWM version of the
## motif; these calculations do not work if any -Inf values are present
examplemotif["pseudocount"] <- 1
# or
examplemotif <- BiocGenerics::normalize(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(list(examplemotif, examplemotif), pvalue = c(0.001, 0.0001))
```

---

motif_rc *Get the reverse complement of a motif.*

Description

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

Usage

```
motif_rc(motifs)
```

Arguments

`motifs`	See `convert_motifs()` for acceptable formats

Value

See `convert_motifs()` for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

Examples

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

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 via **ape**. This can be plotted with `ggtree::ggtree()`.

Usage

```
motif_tree(motifs, layout = "circular", linecol = "family",
  labels = "none", tipsize = "none", legend = TRUE,
  branch.length = "none", db.scores, method = "MPCC",
  use.type = "PPM", min.overlap = 6, tryRC = TRUE,
  min.mean.ic = 0.5, relative_entropy = FALSE, progress = TRUE,
  BP = FALSE, ...)
```

Arguments

motifs	list, dist See `convert_motifs()` for available formats. Alternatively, the resulting comparison matrix from `compare_motifs()`.
layout	character(1) One of c('rectangular', 'slanted', 'fan', 'circular', 'radial', 'equal See `ggtree::ggtree()`.
linecol	character(1) universalmotif slot to use to colour lines (e.g. 'family'). Not available for dist input. 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 `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 c('PCC', 'MPCC', 'EUCL', 'MEUCL', 'SW', 'MSW', 'KL', 'MKL'). See `compare_motifs()`.
use.type	character(1)c('PPM', 'ICM'). The latter allows for taking into account the backgrour ative_entropy = TRUE'). See `compare_motifs()`.
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 less than 1, representing the minimum fraction that the motifs must overlap. See `compare_motifs()`.
tryRC	logical(1) Try the reverse complement of the motifs as well, report the best score. See `compare_motifs()`.
min.mean.ic	numeric(1) Minimum 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. See `compare_motifs()`.

relative_entropy
: logical(1) For ICM calculation. See convert_type().

progress
: logical(1) Show progress of compare_motifs(). Not recommended if BP = TRUE.

BP
: logical(1) Allows the use of **BiocParallel** within compare_motifs(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for comparing large numbers of motifs (>10,000). Furthermore, the behaviour of progress = TRUE is changed if BP = TRUE; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

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

Value

ggplot object.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-0-387-98140-6, http://ggplot2.org.

Yu G, Smith D, Zhu H, Guan Y, Lam TT (2017). "ggtree: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data." *Methods in Ecology and Evolution*, **8**, 28-36. doi: 10.1111/2041210X.12628, http://onlinelibrary.wiley.com/doi/10.1111/2041-210X.12628/abstract.

See Also

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

Examples

```
jaspar <- read_jaspar(system.file("extdata", "jaspar.txt",
                                  package = "universalmotif"))
jaspar.tree <- motif_tree(jaspar, linecol = "none", labels = "name",
                          layout = "rectangular")
```

---

read_cisbp *Import CIS-BP motifs.*

Description

Import CIS-BP formatted motifs. See http://cisbp.ccbr.utoronto.ca/index.php. Assumed to be DNA motifs.

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.

Value

list universalmotif objects.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Weirauch M, Yang A, Albu M, Cote A, Montenegro-Montero A, Drewe P, Najafabadi H, Lambert S, Mann I, Cook K, Zheng H, Goity A, van Bakel H, Lozano J, Galli M, Lewsey M, Huang E, Mukherjee T, Chen X, Reece-Hoyes J, Govindarajan S, Shaulsky G, Walhout A, Bouget F, Ratsch G, Larrondo L, Ecker J, Hughes T (2014). "Determination and inference of eukaryotic transcription factor sequence specificity." *Cell*, **158**, 1431-1443.

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/. Assumed to be DNA motifs.

Usage

```
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, <b2tremblay@uwaterloo.ca>

References

Heinz S, Benner C, Spann N, Bertolino E, Lin Y, Laslo P, Cheng J, Murre C, Singh H, Glass C (2010). "Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities." *Molecular Cell*, **38**, 576-589.

See Also

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

Examples

```
homer <- read_homer(system.file("extdata", "homer.txt",
                                package = "universalmotif"))
```

---

read_jaspar *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, <b2tremblay@uwaterloo.ca>

References

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon J, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas D, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman W, Parcy F, Mathelier A (2018). "JASPAR 2018: update of the open-access database of transcription factor binding profiles and its web framework." *Nucleic Acids Research*, **46**, D260-D266.

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)
```

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')`. 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 seperated.
`headers`	`logical(1)`, `character(1)` Indicating if and how to read names.
`rownames`	`logical(1)` Are there alphabet letters present as rownames?

Value

`list` universalmotif objects.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

See Also

Other read_motifs: read_cisbp, read_homer, read_jaspar, read_meme, read_motifs, read_transfac, read_uniprobe

Examples

```
hocomoco <- system.file("extdata", "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. Both 'full' and 'minimal' formats are supported.

Usage

```
read_meme(file, skip = 0, readsites = 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.

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.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Bailey T, Boden M, Buske F, Frith M, Grant C, Clementi L, Ren J, Li W, Noble W (2009). "MEME SUITE: tools for motif discovery and searching." *Nucleic Acids Research*, **37**, W202-W208.

See Also

Other read_motifs: read_cisbp, read_homer, read_jaspar, read_matrix, read_motifs, read_transfac, read_uniprobe

Examples

```
meme.minimal <- read_meme(system.file("extdata", "meme_minimal.txt",
                                      package = "universalmotif"))
meme.full <- read_meme(system.file("extdata", "meme_full.txt",
                                   package = "universalmotif"))
```

`read_motifs` *Import universalmotif formatted motifs.*

Description

Import motifs created from `write_motifs()`. For optimal storage of `universalmotif` class motifs, consider using `saveRDS()` and `readRDS()`. The `universalmotif` format will not be documented, as realistically the need to generate these manually/elsewhere should be nonexistent.

Usage

```
read_motifs(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, <b2tremblay@uwaterloo.ca>

See Also

Other read_motifs: `read_cisbp`, `read_homer`, `read_jaspar`, `read_matrix`, `read_meme`, `read_transfac`, `read_uniprobe`

`read_transfac` *Import TRANSFAC motifs.*

Description

Import TRANSFAC formatted motifs. Assumed to be DNA motifs, type PCM. See `system.file("extdata", "transfa` for an example motif.

Usage

```
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.

Value

`list` universalmotif objects.

Author(s)

Benjamin Jean-Marie Tremblay, `<b2tremblay@uwaterloo.ca>`

References

Wingender E, Dietze P, Karas H, Knuppel R (1996). "TRANSFAC: A Database on Transcription Factors and Their DNA Binding Sites." *Nucleic Acids Research*, **24**, 238-241.

See Also

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

Examples

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

`read_uniprobe` *Import UNIPROBE motifs.*

Description

Import UNIPROBE formatted motifs. Assumed DNA.

Usage

```
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, <b2tremblay@uwaterloo.ca>

References

Hume M, Barrera L, Gisselbrecht S, Bulyk M (2015). "UniPROBE, update 2015: new tools and content for the online database of protein-binding microarray data on protein-DNA interactions." *Nucleic Acids Research*, **43**, D117-D122.

See Also

Other read_motifs: `read_cisbp`, `read_homer`, `read_jaspar`, `read_matrix`, `read_meme`, `read_motifs`, `read_transfac`

Examples

```
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, run run_meme(). Parameters in run_meme() which are directly taken from the MEME program are tagged with [MEME].

Usage

```
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 overwritting.
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]
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 processx::run().
bin character(1) Location of MEME binary. Alternatively, set this via options(meme.bin = '/path/t

Value

`list` The output file is read with `read_meme()`.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Bailey T, Elkan C (1994). "Fitting a mixture model by expectation maximization to discover motifs in biopolymers." *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, **2**, 28–36.

See Also

`read_meme()`, `create_sequences()`, `shuffle_sequences()`, `processx::run()`

Examples

```
## Not run:
## To check that you are properly linking to the binary:
run_meme()

## End(Not run)
```

---

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

```
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, <b2tremblay@uwaterloo.ca>

See Also

create_sequences(), create_motif(), add_multifreq()

Examples

```
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 = 0.001,
  threshold.type = "pvalue", RC = FALSE, use.freq = 1, verbose = 1,
  progress = TRUE, BP = FALSE)
```

Arguments

motifs	See convert_motifs() for acceptable motif formats.
sequences	XStringSet Sequences to scan. Alphabet should match motif.
threshold	numeric(1) Between 0 and 1. See details.
threshold.type	character(1) One of c('logodds', 'pvalue'). See details.
RC	logical(1) If TRUE, check reverse complement of input sequences.
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 very verbose (3).
progress	logical(1) Show progress. Not recommended if BP = TRUE. Set to FALSE if verbose = 0.
BP	logical(1) Allows for the use of **BiocParallel** within scan_sequences(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for exceptionally large jobs. Keep in mind however that this function will not attempt to limit its memory usage. Furthermore, the behaviour of porgress = TRUE is changed if BP = TRUE; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Details

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, 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.

If threshold.type = 'logodds', then to calculate the minimum allowed score the maximum possible score for a motif is multiplied by the value set by threshold. To determine the maximum possible score a motif (of type PWM), run sum(apply(motif['motif'], 2, max)).

If threshold.type = 'pvalue', then threshold logodds scores are generated using motif_pvalue().

Note: memory usage increases exponentially with increasing k.

Value

data.frame with each row representing one hit; if the input sequences are DNAStringSet or RNAStringSet, then an additional column with the strand is included.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Pagès H, Aboyoun P, Gentleman R, DebRoy S (2018). *Biostrings: Efficient manipulation of biological strings*. R package version 2.48.0.

See Also

add_multifreq(), Biostrings::matchPWM(), enrich_motifs(), motif_pvalue()

Examples

```
## any alphabet can be used
## Not run:
set.seed(1)
alphabet <- paste(c(letters), collapse = "")
motif <- create_motif("hello", alphabet = alphabet)
sequences <- create_sequences(alphabet, seqnum = 1000, seqlen = 100000)
scan_sequences(motif, sequences)

## End(Not run)
```

---

shuffle_motifs *Shuffle motifs by column.*

Description

Given a set of motifs, shuffle the columns between them. 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, they are shuffled together.

Usage

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

Arguments

motifs	See convert_motifs() for acceptable formats.
k	numeric(1) K-let size.
method	character(1) One of c('linear', 'random'). Only relevant if k > 1. See details.
leftovers	character(1) For method = 'random'. One of c('asis', 'first', 'split', 'discard'). See details.

Details

If method = 'linear', then the input positions are split linearly every k columns after which they are shuffled. If method = random, then sets of k-columns are chosen randonmly before being shuffled. This leaves leftover column islands smaller than k; these can be left asis, placed first, split between the beginning and the end, or discarded. See shuffle_motifs().

Value

Motifs. See convert_motifs() for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@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 = "linear",
  leftovers = "asis", progress = FALSE, BP = FALSE)
```

Arguments

sequences	XStringSet For method = 'markov', DNAStringSet and RNAStringSet only.
k	numeric(1) K-let size.
method	character(1) One of c('markov', 'linear', 'random'). Only relevant is k > 1. See details.
leftovers	character(1) For method = 'random'. One of c('asis', 'first', 'split', 'discard'). See details.
progress	logical(1) Show progress. Not recommended if BP = TRUE.
BP	logical(1) Allows the use of **BiocParallel** within shuffle_sequences(). See BiocParallel::register() to change the default backend. Setting BP = TRUE is only recommended for large jobs (such as shuffling billions of letters). Furthermore, the behaviour of progress = TRUE is changed if BP = TRUE; the default **BiocParallel** progress bar will be shown (which unfortunately is much less informative).

Details

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) and Altschul and Erickson (1985) for a discussion on the topic.

If method = 'linear', then the input sequences are split linearly every k letters; for example, for k = 3 'ACAGATAGACCC' becomes 'ACA GAT AGA CCC'; afterwhich these 3-lets are shuffled randomly. If method = 'random', then k-lets are picked from the sequence completely randomly. This however can leave 'leftover' letters, where lone letter islands smaller than k are left. There are a few options provided to deal with these: leftovers = 'asis' will leave these letter islands in place; leftovers = 'first' will place these letters at the beginning of the sequence; leftovers = 'split' will place half of the leftovers at the beginning and end of the sequence; leftovers = 'discard' simply gets rid of the leftovers.

Do note however, that the method parameter is only relevant for k > 1. For this, a simple sample call is performed.

Value

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

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Altschul SF, Erickson BW (1985). "Significance of Nucleotide Sequence Alignments: A Method for Random Sequence Permutation That Preserves Dinucleotide and Codon Usage." *Molecular Biology and Evolution*, **2**, 526-538.

Fitch WM (1983). "Random sequences." *Journal of Molecular Biology*, **163**, 171-176.

See Also

create_sequences(), scan_sequences(), enrich_motifs(), shuffle_motifs()

Examples

```
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

```
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, <b2tremblay@uwaterloo.ca>

See Also

`create_motif()`

Examples

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

`trim_motifs` *Trim motifs.*

Description

Remove edges of motifs with low information content.

Usage

```
trim_motifs(motifs, min.ic = 0.25)
```

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.

Value

Motifs See convert_motifs() for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

See Also

create_motif(), convert_type()

Examples

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

---

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,ANY,ANY,ANY'
x[i]

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

## S4 method for signature 'universalmotif'
initialize(.Object, name, altname, family,
  organism, motif, alphabet = "DNA", type, icscore, nsites,
  pseudocount = 0.8, bkg, bkgsites, consensus, strand = "+-", pval,
  qval, eval, multifreq, extrainfo)

## 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)

## S4 method for signature 'universalmotif'
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.
altname	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', 'custom'), 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 apearing in PWM matrices.
bkg	numeric Must sum to 1 and be equal in length to the alphabet length. If missing, assumes a uniform background.
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.
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)

icscore numeric(1) Generated automatically.

nsites numeric(1)

pseudocount numeric(1)

bkg numeric Length equal to number of letters in alphabet.

bkgsites numeric(1)

consensus character Generated automatically.

strand character(1)

pval numeric(1)

qval numeric(1)

eval numeric(1)

multifreq list

extrainfo character

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

Examples

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

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

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

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

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

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

## rowMeans
## Calculate motif rowMeans.
rowMeans(motif)

## colMeans
## Calculate motif colMeans.
colMeans(motif)

## colSums
## Calculate motif colSums
colSums(motif)

## rowSums
## Calculate motif rowSums.
rowSums(motif)

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

```
## ncol
## Count motif columns.
ncol(motif)

## colnames
## Get motif colnames.
colnames(motif)

## rownames
## Get motif rownames.
rownames(motif)

## cbind
## Bind motifs together to create a new motif.
cbind(motif, motif2)
```

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

Description

universalmotif: Import, Modify and Export Motifs with R

Details

A collection of utility functions for working with motifs.

`view_motifs` *Plot motif logos.*

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

```
view_motifs(motifs, use.type = "ICM", method = "MPCC", tryRC = TRUE,
  min.overlap = 6, min.mean.ic = 0.5, relative_entropy = FALSE,
  normalise.scores = FALSE, ...)
```

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 `c('PCC', 'MPCC', 'EUCL', 'MEUCL', 'SW', 'MSW', 'KL', 'MKL')`. See `compare_motifs()`.

`tryRC` `logical(1)` Check if motif reverse complement leads to a better alignment. See `compare_motifs()`.

`min.overlap` `numeric(1)` Minimum alignment overlap between motifs. If `min.overlap < 1`, this represents the minimum fraction between the two motifs during alignment. See `compare_motifs()`.

`min.mean.ic` `numeric(1)` Minimum 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. See `compare_motifs()`.

`relative_entropy` `logical(1)` For ICM calculation. See `convert_type()`.

`normalise.scores` `logical(1)` Favour alignments which leave fewer unaligned positions. See `compare_motifs()`.

`...` Addtional options for `ggseqlogo::geom_logo()`.

Details

Since the **ggseqlogo** package can only plot individual characters and not strings, plotting the `multifreq` slot is not supported. See the `examples` section for plotting the `multifreq` slot using the **Logolas** package.

Value

A ggplot object.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Dey KK, Xie D, Stephens M (2017). "A new sequence logo plot to highlight enrichment and depletion." *bioRxiv, p.226597*.

Wagih O (2017). *ggseqlogo: A 'ggplot2' Extension for Drawing Publication-Ready Sequence Logos*. R package version 0.1, https://CRAN.R-project.org/package=ggseqlogo.

See Also

`compare_motifs()`, `add_multifreq()`

Examples

```
## plotting multifreq motifs:
## Not run:
  motif <- create_motif()
  motif <- add_multifreq(motif, sample_sites(motif))
  Logolas::logomaker(motif["multifreq"][[as.character(2)]], type = "Logo",
                     color_type = "per_symbol")

## 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/.

Usage

```
write_homer(motifs, file, logodds_threshold = 0.6)
```

Arguments

motifs	See convert_motifs() for acceptable formats.
file	character(1) File name.
logodds_threshold	numeric Stringency required for HOMER to match a motif. See scan_sequences().

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Heinz S, Benner C, Spann N, Bertolino E, Lin Y, Laslo P, Cheng J, Murre C, Singh H, Glass C (2010). "Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities." *Molecular Cell*, **38**, 576-589.

See Also

read_homer()

Other write_motifs: write_jaspar, write_matrix, write_meme, write_motifs, write_transfac

Examples

```
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)
```

Arguments

motifs	See convert_motifs() for acceptable formats.
file	character(1) File name.

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon J, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas D, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman W, Parcy F, Mathelier A (2018). "JASPAR 2018: update of the open-access database of transcription factor binding profiles and its web framework." *Nucleic Acids Research*, **46**, D260-D266.

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())
```

write_matrix *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)
```

Arguments

motifs	See convert_motifs() for acceptable formats.
file	character(1) File name.
positions	character(1) One of c('columns', 'rows').
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.
headers	logical(1), character(1) Indicating if and how to write names.

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@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 = 4, bkg, strand)
```

Arguments

motifs	See convert_motifs() for acceptable formats.
file	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 "+ -"

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Bailey T, Boden M, Buske F, Frith M, Grant C, Clementi L, Ren J, Li W, Noble W (2009). "MEME SUITE: tools for motif discovery and searching." *Nucleic Acids Research*, **37**, W202-W208.

See Also

read_meme()

Other write_motifs: write_homer, write_jaspar, write_matrix, write_motifs, write_transfac

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()`. The `universalmotif` format will not be documented, as realistically the need to generate these manually/elsewhere should be nonexistent.

Usage

```
write_motifs(motifs, file, minimal = FALSE, multifreq = TRUE)
```

Arguments

`motifs`	See `convert_motifs()` for acceptable formats.
`file`	`character(1)` File name.
`minimal`	`logical(1)` Only write essential motif information.
`multifreq`	`logical(1)` Write `multifreq` slot, if present.

Value

`NULL`, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@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)
```

Arguments

`motifs`	See `convert_motifs()` for acceptable formats.
`file`	`character(1)` File name.

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <b2tremblay@uwaterloo.ca>

References

Wingender E, Dietze P, Karas H, Knuppel R (1996). "TRANSFAC: A Database on Transcription Factors and Their DNA Binding Sites." *Nucleic Acids Research*, **24**, 238-241.

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