Package 'universalmotif'

October 17, 2020

Title Import, Modify, and Export Motifs with R

Version 1.6.4

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, stats, utils, MASS, ggplot2, ape, ggtree, ggseqlogo, yaml, Rcpp, Rdpack (>= 0.7), Biostrings, BiocGenerics, processx, S4Vectors, rlang

Suggests spelling, knitr, bookdown, TFBSTools, rmarkdown, MotifDb, testthat, Logolas, BiocParallel, seqLogo, motifStack, dplyr

Enhances MotIV, PWMEnrich, rGADEM, motifRG

LinkingTo Rcpp, RcppThread

RdMacros Rdpack

VignetteBuilder knitr

biocViews MotifAnnotation, MotifDiscovery, DataImport, GeneRegulation

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE, old_usage = TRUE)

Language en-GB

Collate 'RcppExports.R' 'add_multifreq.R' 'compare_motifs.R' 'universalmotif-class.R' 'convert_motifs.R' 'convert_type.R' 'create_motif.R' 'create_sequences.R' 'data.R' 'enrich_motifs.R' 'filter_motifs.R' 'get_bkg.R' 'make_DBscores.R' 'merge_motifs.R' 'motif_clusters.R' 'motif_peaks.R' 'motif_pvalue.R' 'motif_rc.R' 'motif_tree.R' 'read_cisbp.R' 'read_homer.R' 'read_jaspar.R' 'read_matrix.R' 'read_meme.R' 'read_motifs.R' 'read_transfac.R'

'read_uniprobe.R' 'run_meme.R' 'sample_sites.R'
'scan_sequences.R' 'shuffle_motifs.R' 'shuffle_sequences.R'
'switch_alph.R' 'trim_motifs.R' 'universalmotif-methods.R'
'universalmotif.R' 'utils-internal.R' 'utils-motif.R'
'utils-sequence.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_11

git_last_commit 91b2ec6

git_last_commit_date 2020-09-17

Date/Publication 2020-10-16

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

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. See the "Motif import, export, and manipulation" vignette for more information.

Usage

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

Arguments

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

Details

See scan_sequences() for more info on scanning parameters.

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

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

This 'multifreq' representation is only really useful within the **universalmotif** environment. Despite this, if you wish it can be preserved in text using write_motifs().

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. The bkg slot is also expanded with corresponding higher order probabilities if add.bkg = TRUE.

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 -dna.

Usage

```
ArabidopsisMotif
```

Format

universalmotif

ArabidopsisPromoters *Arabidopsis promoters as a* DNAStringSet.

Description

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

Usage

```
ArabidopsisPromoters
```

Format

DNAStringSet

compare_motifs *Compare motifs.*

Description

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

Usage

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

Arguments

motifs	See convert_motifs() for acceptable motif formats.
compare.to	numeric If missing, compares all motifs to all other motifs. Otherwise compares all motifs to the specified motif(s).
db.scores	data.frame or DataFrame. See details.
use.freq	numeric(1). For comparing the multifreq slot.
use.type	character(1) One of 'PPM' and 'ICM'. The latter allows for taking into account the background frequencies if relative_entropy = TRUE. Note that 'ICM' is not allowed when method = c("ALLR","ALLR_LL").
method	character(1) One of PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
tryRC	logical(1) Try the reverse complement of the motifs as well, report the best score.
min.overlap	numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
min.mean.ic	numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs.
min.position.ic	numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.
relative_entropy	logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic. See convert_type().
normalise.scores	logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
max.p	numeric(1) Maximum P-value allowed in reporting matches. Only used if compare.to is set.
max.e	numeric(1) Maximum E-value allowed in reporting matches. Only used if compare.to is set. The E-value is the P-value multiplied by the number of input motifs times two.
nthreads	numeric(1) Run compare_motifs() in parallel with nthreads threads. nthreads = 0 uses all available threads.
score.strat	character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.
output.report	character(1) Provide a filename for compare_motifs() to write an html ouput report to. The top matches are shown alongside figures of the match alignments. This requires the knitr and rmarkdown packages. (Note: still in development.)

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

Details

The following metrics are available:

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

Comparisons are calculated between two motifs at a time. All possible alignments are scored, and the best score is reported. In an alignment scores are calculated individually between columns. How those scores are combined to generate the final alignment scores depends on `score.strat`.

See the "Motif comparisons and P-values" vignette for a description of the various metrics. Note that `PCC`, `WPCC`, `SW`, `ALLR`, `ALLR_LL` and `BHAT` are similarities; higher values mean more similar motifs. For the remaining metrics, values closer to zero represent more similar motifs.

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

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

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

Value

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

Author(s)

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

References

Bhattacharyya A (1943). "On a measure of divergence between two statistical populations defined by their probability distributions." *Bulletin of the Calcutta Mathematical Society*, **35**, 99–109.

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

Hellinger E (1909). "Neue Begrundung der Theorie quadratischer Formen von unendlichvielen Veranderlichen." *Journal fur die reine und angewandte Mathematik*, **136**, 210–271.

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon JA, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas DJ, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman WW, 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.

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

Itakura F, Saito S (1968). "Analysis synthesis telephony based on the maximum likelihood method." In *6th International Congress on Acoustics*, C–17.

Mahony S, Auron PE, Benos PV (2007). "DNA Familial Binding Profiles Made Easy: Comparison of Various Motif Alignment and Clustering Strategies." *PLoS Computational Biology*, **3**.

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

Wang T, Stormo GD (2003). "Combining phylogenetic data with co-regulated genes to identify motifs." *Bioinformatics*, **19**, 2369–2380.

See Also

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

Examples

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

motif3 <- create_motif(name = "3")
motif4 <- create_motif(name = "4")
motifs <- c(motif1, motif2, motif3, motif4)
## Compare motif "2" to all the other motifs:
if (R.Version()$arch != "i386") {
compare_motifs(motifs, compare.to = 2, max.p = 1, max.e = Inf)
}
```

convert_motifs *Convert motif class.*

Description

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

Usage

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

## S4 method for signature '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.

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. See the "Introduction to sequence motifs" vignette for details.

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 appearing in PWM matrices. If missing, the pseudocount stored in the universalmotif 'pseudocount' slot will be used.
`nsize_correction`	`logical(1)` If true, the ICM at each position will be corrected to account for small sample sizes. Only used if `relative_entropy = FALSE`.
`relative_entropy`	`logical(1)` If true, the ICM will be calculated as relative entropy. See details.

Details

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-likelihood 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"))

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

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

---

create_motif *Create a motif.*

Description

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

Usage

```
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, numeric, matrix, XStringSet, or missing.
alphabet	character(1) One of c('DNA','RNA','AA'), or a combined string representing the letters. If no alphabet is provided then it will try and guess the alphabet from the input.
type	character(1) One of c('PCM','PPM','PWM','ICM').

name	character(1) Motif name.
pseudocount	numeric(1) Correction to be applied to prevent -Inf from appearing in PWM matrices. Defaults to 0.
bkg	numeric A vector of probabilities, each between 0 and 1. If higher order backgrounds are provided, then the elements of the vector must be named. If unnamed, then the order of probabilities must be in the same order as the alphabetically sorted sequence alphabet.
nsites	numeric(1) Number of sites the motif was constructed from. If blank, then create_motif() will guess the appropriate number if possible. To prevent this, provide nsites = numeric().
altname	character(1) Alternate motif name.
family	character(1) Transcription factor family.
organism	character(1) Species of origin.
bkgsites	numeric(1) Total number of sites used to find the motif.
strand	character(1) Whether the motif is specific to a certain strand.
pval	numeric(1) P-value associated with motif.
qval	numeric(1) Adjusted P-value associated with motif.
eval	numeric(1) E-value associated with motif.
extrainfo	character Any other extra information, represented as a named character vector.
add.multifreq	numeric If the motif is created from a set of sequences, then the add_multifreq() function can be run at the same time (with RC = FALSE).

Details

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

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

See the examples section for more info on motif creation.

Value

universalmotif object.

Methods (by class)

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

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 or
# a single string is used as input
DNA.motif <- create_motif("TTTTTTT", nsites = 10)

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

# Be careful about setting nsites when using ambiguity letters!
DNA.motif <- create_motif("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 = "QWER")
# Specify custom alphabet
custom.motif <- create_motif("QWER", alphabet = "QWERASDF")

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

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

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

## 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, alphabet = "QWER")

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

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

# Matrices can also be used as input
mat.ppm <- matrix(c(0.1, 0.1, 0.1, 0.1,
                    0.5, 0.5, 0.5, 0.5,
                    0.1, 0.1, 0.1, 0.1,
                    0.3, 0.3, 0.3, 0.3),
                    nrow = 4, byrow = TRUE)

DNA.motif <- create_motif(mat.ppm, alphabet = "DNA", type = "PPM")

##### Create random motifs

# These are generated as PPMs with 10 positions

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, freqs,
  nthreads = 1, rng.seed = sample.int(10000, 1))
```

Arguments

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

Value

`XStringSet` The returned sequences are *unnamed*.

Author(s)

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

See Also

create_motif(), shuffle_sequences()

Examples

```
## Create DNA sequences with slightly increased AT content:
sequences <- create_sequences(freqs = c(A=0.3, C=0.2, G=0.2, T=0.3))
## Create custom sequences:
sequences.QWER <- create_sequences("QWER")
## You can include non-alphabet characters 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. See the "Sequence manipulation and scanning" vignette.

Usage

```
enrich_motifs(motifs, sequences, bkg.sequences, max.p = 1e-05,
  max.q = 1e-05, max.e = 0.001, qval.method = "fdr", threshold = 0.001,
  threshold.type = "pvalue", verbose = 0, RC = FALSE, use.freq = 1,
  shuffle.k = 2, shuffle.method = "euler", return.scan.results = FALSE,
  nthreads = 1, rng.seed = sample.int(10000, 1), motif_pvalue.k = 8)
```

Arguments

motifs	See convert_motifs() for acceptable motif formats.
sequences	XStringSet Sequences to scan. Alphabet should match motif.
bkg.sequences	XStringSet Optional. If missing, shuffle_sequences() is used to create background sequences from the input sequences.
max.p	numeric(1) P-value threshold.
max.q	numeric(1) Adjusted P-value threshold. This is only useful if multiple motifs are being enriched for.
max.e	numeric(1). The E-value is calculated by multiplying the adjusted P-value with the number of input motifs times two (McLeay and Bailey 2010).
qval.method	character(1) See stats::p.adjust().
threshold	numeric(1) See details.
threshold.type	character(1) One of c('logodds','logodds.abs','pvalue'). See details.
verbose	numeric(1) 0 for no output, 4 for max verbosity.
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().

shuffle.k numeric(1) The k-let size to use when shuffling input sequences. Only used if no background sequences are input. See shuffle_sequences().

shuffle.method character(1) One of c('euler','markov','linear'). See shuffle_sequences().

return.scan.results
logical(1) Return output from scan_sequences(). For large jobs, leaving this as FALSE can save a small amount time by preventing construction of the complete results data.frame from scan_sequences().

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

rng.seed numeric(1) Set random number generator seed. Since shuffling can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Each individual sequence in an XStringSet object will be given the following seed: rng.seed * index. See shuffle_sequences().

motif_pvalue.k numeric(1) Control motif_pvalue() approximation. See motif_pvalue().

Details

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

See scan_sequences() for more info on scanning parameters.

Value

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

Author(s)

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

References

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

examplemotif *Example motif in* universalmotif *format.*

Description

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

Usage

```
examplemotif
```

Format

universalmotif

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

Description

A simple DNA motif with a non-empty `multifreq` slot. To recreate to this motif: `add_multifreq(examplemotif,DNAS`

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, extrainfo)
```

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.
extrainfo	character Named character vector of items that must be present in motif extrainfo slots.

Value

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

Author(s)

Benjamin Jean-Marie Tremblay, <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:
## Not run:
library(MotifDb)
motifs <- convert_motifs(MotifDb)
motifs <- filter_motifs(motifs, organism = c("Athaliana", "Mmusculus"),
                        extrainfo = c("dataSource" = "cisbp_1.02"))

## End(Not run)
```

get_bkg *Calculate sequence background.*

Description

For a set of input sequences, calculate the overall sequence background for any k-let size. For very large sequences, this is only recommended for non-DNA/RNA sequences: otherwise use the much faster and more efficient `Biostrings::oligonucleotideFrequency()`.

Usage

```
get_bkg(sequences, k = 1:3, as.prob = TRUE, pseudocount = 0,
  alphabet = NULL, to.meme = NULL, RC = FALSE, list.out = TRUE,
  nthreads = 1)
```

Arguments

sequences	`XStringSet` Input sequences. Note that if multiple sequences are present, the results will be combined into one.
k	`integer` Size of k-let. Background can be calculated for any k-let size.
as.prob	`logical(1)` Whether to return k-let counts or probabilities.
pseudocount	`integer(1)` Add a count to each possible k-let. Prevents any k-let from having 0 or 1 probabilities.
alphabet	`character(1)` Provide a custom alphabet to calculate a background for. If `NULL`, then standard letters will be assumed for DNA, RNA and AA sequences, and all unique letters found will be used for `BStringSet` type sequences.
to.meme	If not `NULL`, then `get_bkg()` will return the sequence background in MEME Markov Background Model format. Input for this argument will be used for `cat(...,file = to.meme)` within `get_bkg()`. See `http://meme-suite.org/doc/bfile-format.html` for a description of the format.
RC	`logical(1)` Calculate the background of the reverse complement of the input sequences as well. Only valid for DNA/RNA.
list.out	`logical(1)` Return background frequencies as list, with an entry for each `k`. If `FALSE`, return a single vector.
nthreads	`numeric(1)` Run `get_bkg()` in parallel with `nthreads` threads. `nthreads = 0` uses all available threads. Note that no speed up will occur for jobs with only a single sequence.

Value

If `to.meme = NULL` and `list.out = TRUE`: a list with each entry being a named numeric vector for every element in `k`. If `to.meme = NULL` and `list.out = FALSE`: a named numeric vector. Otherwise: `NULL`, invisibly.

Author(s)

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

References

Bailey TL, 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

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

Examples

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

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

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

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. The scores are organized in a `DataFrame`. In this `DataFrame` is the location and scale of scores resulting from a statistical distribution using the the comparisons of JASPAR2018 CORE motifs with randomized motifs of the specified `subject` and `target` motif length. Created using `make_DBscores()` from universalmotif v1.4.0. The parameters used can be seen via `S4Vectors::metadata(JASPAR2018_CORE_DBSCORES)`.

Usage

```
JASPAR2018_CORE_DBSCORES
```

Format

`DataFrame` with function args in the `metadata` slot.

make_DBscores *Create P-value databases.*

Description

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

Usage

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

Arguments

db.motifs	list Database motifs.
method	character(1) One of PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
shuffle.db	logical(1) Deprecated. Does nothing. generate random motifs with create_motif().
shuffle.k	numeric(1) See shuffle_motifs().
shuffle.method	character(1) See shuffle_motifs().
rand.tries	numeric(1) Approximate number of comparisons to perform for every combination of widths.
widths	numeric Motif widths to use in P-value database calculation.
min.position.ic	numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.
normalise.scores	logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
min.overlap	numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
min.mean.ic	numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs.
progress	logical(1) Show progress.

nthreads	numeric(1) Run compare_motifs() in parallel with nthreads threads. nthreads = 0 uses all available threads.
tryRC	logical(1) Try the reverse complement of the motifs as well, report the best score.
score.strat	character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

Details

See compare_motifs() for more info on comparison parameters.

To replicate the internal **universalmotif** DB scores, run make_DBscores() with the default settings. Note that this will be a slow process.

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

Value

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

Author(s)

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

References

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon JA, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas DJ, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman WW, 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

compare_motifs()

Examples

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

## End(Not run)
```

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. Only 0-order background probabilities will be kept. Motifs are merged one at a time, starting with the first entry in the list.

Usage

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

Arguments

motifs	See convert_motifs() for acceptable motif formats.
method	character(1) One of PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
use.type	character(1) One of 'PPM' and 'ICM'. The latter allows for taking into account the background frequencies if relative_entropy = TRUE. Note that 'ICM' is not allowed when method = c("ALLR","ALLR_LL").
min.overlap	numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
min.mean.ic	numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs.
tryRC	logical(1) Try the reverse complement of the motifs as well, report the best score.
relative_entropy	logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic. See convert_type().
normalise.scores	logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
min.position.ic	numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.
score.strat	character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean":

weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.

`new.name` `character(1)`, `NULL` Instead of collapsing existing names (if `NULL`), assign a new for the merged motif.

Details

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

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

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

Value

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

Author(s)

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

See Also

`compare_motifs()`

Examples

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

## End(Not run)

# Using ALLR or KL will add a pseudocount to the input motifs. Compare:
m1 <- create_motif("TTAAACCCC", name = "1")
m2 <- create_motif("AACC", name = "2")
m3 <- create_motif("AACCCCGG", name = "3")
view_motifs(merge_motifs(c(m1, m2, m3), method = "PCC"))
view_motifs(merge_motifs(c(m1, m2, m3), method = "ALLR"))
```

---

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

Description

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

Usage

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

Arguments

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

Details

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

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

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

Value

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

Author(s)

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

References

Wand M (2015). *KernSmooth: Functions for Kernel Smoothing Supporting Wand and Jones (1995)*. R package version 2.23-15, https://CRAN.R-project.org/package=KernSmooth.

See Also

scan_sequences()

Examples

```
data(ArabidopsisMotif)
data(ArabidopsisPromoters)
if (R.Version()$arch != "i386") {
hits <- scan_sequences(ArabidopsisMotif, ArabidopsisPromoters, RC = FALSE)
res <- motif_peaks(as.vector(hits$start), 1000, 50)
# View plot:
res$Plot

# The raw plot data can be found in:
res$Plot$data
}
```

---

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 = 8,
  nthreads = 1, rand.tries = 10, rng.seed = sample.int(10000, 1))
```

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, retrieves the background from the motif bkg slot. Note that this only influences calculating p-values from an input score; calculating a score from an input p-value currently assumes a uniform background.
use.freq	numeric(1) By default uses the regular motif matrix; otherwise uses the corresponding multifreq matrix. Max is 3.
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 (nearly) exact. The default, 8, is recommended to those looking for a good tradeoff between speed and accuracy for jobs requiring repeated calculations.
nthreads	numeric(1) Run motif_pvalue() in parallel with nthreads threads. nthreads = 0 uses all available threads.

`rand.tries` `numeric(1)` When `ncol(motif) < k`, an approximation is used. This involves randomly approximating the overall motif score distribution. To increase accuracy, the distribution is approximated `rand.tries` times and the final scores averaged.

`rng.seed` `numeric(1)` In order to allow `motif_pvalue()` to perform C++ level parallelisation, it must work independently from R. This means it cannot communicate with R to get/set the R RNG state. To get around this, the RNG seed used by the C++ function can be set with `rng.seed`. To make sure each thread gets a different seed however, the seed is multiplied with the iteration count. For example: when working with two motifs, the second motif gets the following seed: `rng.seed * 2`. The default is to pick a random number as chosen by `sample()`, which effectively makes `motif_pvalue()` dependent on the R RNG state.

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 even this calculation becomes impractical. To get around this, the p-value calculation can be approximated.

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

To calculate a score from a P-value, all possible scores are calculated and the `(1 -pvalue)` nth percentile score returned. When `k < ncol(motif)`, the complete set of scores is instead approximated by randomly adding up all possible scores from each subset. It is important to keep in mind that no consideration is given to background frequencies in the score calculator. Note that this approximation can actually be potentially quite expensive at times and even slower than the exact version; for jobs requiring lots of repeat calculations, a bit of benchmarking beforehand can be useful to find the optimal settings.

To get an idea as to how the score calculator works (without approximation), try the following code with your motif (be careful with longer motifs):

```
quantile(get_scores(motif),probs = 0.99)
```

Value

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

Author(s)

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

References

Hartmann H, Guthohrlein EW, Siebert M, Soding SLJ (2013). "P-value-based regulatory motif discovery using positional weight matrices." *Genome Research*, **23**, 181–194.

See Also

motif_score()

Examples

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

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

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

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

---

motif_rc *Get the reverse complement of a DNA or RNA 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, ignore.alphabet = FALSE)
```

Arguments

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

Value

See convert_motifs() for available output formats.

Author(s)

Benjamin Jean-Marie Tremblay, <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. This can be plotted with ggtree::ggtree(). The purpose of this function is simply to combine the compare_motifs() and ggtree::ggtree() steps into one. For more control over tree creation, it is recommend to do these steps separately. See the "Motif comparisons and P-values" vignette for such a workthrough.

Usage

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

Arguments

motifs	list, dist See convert_motifs() for available formats. Alternatively, the resulting comparison matrix from compare_motifs() (run as.dist(results) beforehand; if the comparison was performed with a similarity metric, make sure to convert to distances first).
layout	character(1) One of c('rectangular','slanted','fan','circular','radial','equal_ang See ggtree::ggtree().

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

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

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

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

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

db.scores — data.frame See compare_motifs().

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

use.type — character(1)c('PPM', 'ICM'). The latter allows for taking into account the background frequencies (only if relative_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 between 0 and 1, representing the minimum fraction that the motifs must overlap.

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

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

min.mean.ic — numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs.

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

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

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

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

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

Details

See compare_motifs() for more info on comparison parameters.

Value

ggplot object.

Author(s)

Benjamin Jean-Marie Tremblay, <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")

## Not run:
## When inputting a dist object, the linecol and tipsize options are
## not available. To add these manually:

library(MotifDb)
library(ggtree)
library(ggplot2)

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

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

tree <- tree %<+% annotations +
          geom_tippoint(aes(size = icscore)) +
          aes(colour = family) +
          theme(legend.position = "right",
                legend.title = element_blank())

## End(Not run)
```

`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.

Details

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

Value

`list` universalmotif objects.

Author(s)

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

References

Weirauch MT, Yang A, Albu M, Cote AG, Montenegro-Montero A, Drewe P, Najafabadi HS, Lambert SA, Mann I, Cook K, Zheng H, Goity A, van Bakel H, Lozano JC, Galli M, Lewsey MG, Huang E, Mukherjee T, Chen X, Reece-Hoyes JS, Govindarajan S, Shaulsky G, Walhout AJ, Bouget FY, Ratsch G, Larrondo LF, Ecker JR, Hughes TR (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 YC, Laslo P, Cheng JX, Murre C, Singh H, Glass CK (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 JA, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas DJ, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman WW, 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 separated.
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, readsites.meta = FALSE)
```

Arguments

file	character(1) File name.
skip	numeric(1) If not zero, will skip however many desired lines in the file before starting to read.
readsites	logical(1) If TRUE, the motif sites will be read as well.
readsites.meta	logical(1) If readsites = TRUE, then additionally read site positions and P-values.

Details

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

Furthermore, note that DNA-LIKE, RNA-LIKE, and AA-LIKE alphabets will be treated as DNA, RNA, and AA alphabets respectively. Custom alphabets are currently not supported.

Value

list universalmotif objects. If readsites = TRUE, a list comprising of a sub-list of motif objects and a sub-list of motif sites will be returned. If readsites.meta = TRUE, then two additional list items will be present, one containing site positions and P-values, and another containing combined sequence p-values.

Author(s)

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

References

Bailey TL, Boden M, Buske FA, Frith M, Grant CE, Clementi L, Ren J, Li WW, Noble WS (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"))
## Get numeric p-value:
log_string_pval(meme.minimal[[1]]["extrainfo"]["eval.string"])
```

`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()`. Currently the `universalmotif` format is YAML-based, but this is subject to change.

Usage

```
read_motifs(file, skip = 0, progress = FALSE, BP = 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.
`progress`	`logical(1)` Show progress.
`BP`	`logical(1)` Allows for the use of **BiocParallel** within `read_motifs()`. See `BiocParallel::register()` to change the default backend.

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 MA, Barrera LA, Gisselbrecht SS, Bulyk ML (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 over-writting.
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/to/meme/bin').

Value

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

Author(s)

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

References

Bailey TL, 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 = 0,
  nthreads = 1, motif_pvalue.k = 8)
```

Arguments

motifs	See convert_motifs() for acceptable motif formats.
sequences	XStringSet Sequences to scan. Alphabet should match motif.
threshold	numeric(1) See details.
threshold.type	character(1) One of c('logodds','logodds.abs','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 verbose (3).
nthreads	numeric(1) Run scan_sequences() in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with only a single motif and sequence.
motif_pvalue.k	numeric(1) Control motif_pvalue() approximation. See motif_pvalue().

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 max possible score for a motif is multiplied by the value set by threshold. To determine the maximum possible scores a motif (of type PWM), run motif_score(motif,1). If threshold.type = 'pvalue',

then threshold logodds scores are generated using motif_pvalue(). Finally, if threshold.type = 'logodds.abs', then the exact values provided will be used as thresholds.

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

Value

DataFrame with each row representing one hit. If the input sequences are DNAStringSet or RNAStringSet, then an additional column with the strand is included. Function args are stored in the metadata slot.

Author(s)

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

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)

## Sequence masking:
if (R.Version()$arch != "i386") {
library(Biostrings)
data(ArabidopsisMotif)
data(ArabidopsisPromoters)
seq <- ArabidopsisPromoters[[1]]  # Only works for XString, not XStringSet
seq <- mask(seq, pattern = "AAAA")  # MaskedDNAString class
seq <- injectHardMask(seq, letter = "+")  # Recover XString
seq <- DNAStringSet(seq)  # scan_sequences() needs XStringSet
scan_sequences(ArabidopsisMotif, seq)
# A warning regarding the presence of non-standard letters will be given,
# but can be safely ignored in this case.

# If you'd like to do it to a whole DNAStringSet object:
seq <- ArabidopsisPromoters
for (i in 1:length(ArabidopsisPromoters)) {
  seq[[i]] <- injectHardMask(mask(seq[[i]], pattern = "AAAA"), letter = "+")
}
scan_sequences(ArabidopsisMotif, seq)
}
```

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")
```

Arguments

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

Value

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

Author(s)

Benjamin Jean-Marie Tremblay, <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 = "euler", nthreads = 1,
  rng.seed = sample.int(10000, 1))
```

Arguments

sequences	XStringSet Set of sequences to shuffle. Works with any set of characters.
k	numeric(1) K-let size.
method	character(1) One of c('euler','markov','linear'). Only relevant if k > 1. See details.
nthreads	numeric(1) Run shuffle_sequences() in parallel with nthreads threads. nthreads = 0 uses all available threads. Note that no speed up will occur for jobs with only a single sequence.
rng.seed	numeric(1) Set random number generator seed. Since shuffling can occur simultaneously in multiple threads using C++, it cannot communicate with the regular R random number generator state and thus requires an independent seed. Each individual sequence in an XStringSet object will be given the following seed: rng.seed * index. The default is to pick a random number as chosen by sample(), which effectively is making shuffle_sequences() dependent on the R RNG state.

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) for a discussion on the topic.

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

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

Do note however, that the method parameter is only relevant for k > 1. For k = 1, 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.

Propp JG, Wilson DW (1998). "How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph." *Journal of Algorithms*, **27**, 170–217.

See Also

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

Examples

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

---

switch_alph *Switch between DNA and RNA alphabets.*

Description

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

Usage

```
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. Currently does not trim `multifreq` representations.

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'
x[i]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 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'), or a combined string representing the letters.
type	character(1) One of c('PCM','PPM','PWM','ICM').
icscore	numeric(1) Total information content. Automatically generated.
nsites	numeric(1) Number of sites the motif was constructed from.
pseudocount	numeric(1) Correction to be applied to prevent -Inf from appearing in PWM matrices.
bkg	numeric A vector of probabilities, each between 0 and 1. If higher order backgrounds are provided, then the elements of the vector must be named.
bkgsites	numeric(1) Total number of sites used to find the motif.
consensus	character(1) Consensus string. Automatically generated for 'DNA', 'RNA', and 'AA' alphabets.
strand	character(1) Whether the motif is specific to a certain strand.
pval	numeric(1) P-value associated with motif.
qval	numeric(1) Adjusted P-value associated with motif.
eval	numeric(1) E-value associated with motif.
multifreq	list See add_multifreq().
extrainfo	character Any other extra information, represented as a named character vector.
gapinfo	universalmotif_gapped(1) Gapped motif information.
object	universalmotif Motif.
select	numeric Columns to keep.
...	universalmotif Motifs.
deparse.level	Unused.

Value

A motif object of class universalmotif.

Slots

`name` character(1)

`altname` character(1)

`family` character(1)

`organism` character(1)

`motif` matrix

`alphabet` character(1)

`type` character(1)

`icscore` numeric(1) Generated automatically.

`nsites` numeric(1)

`pseudocount` numeric(1)

`bkg` numeric 0-order probabilities must be provided for all letters.

`bkgsites` numeric(1)

`consensus` character Generated automatically.

`strand` character(1)

`pval` numeric(1)

`qval` numeric(1)

`eval` numeric(1)

`multifreq` list

`extrainfo` character

`gapinfo` universalmotif_gapped(1)

Author(s)

Benjamin Jean-Marie Tremblay, <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

A collection of utility functions for working with motifs.

utilities *Utility functions.*

Description

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

Author(s)

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

See Also

utils-motif, utils-sequence

utils-motif *Motif-related utility functions.*

Description

Motif-related utility functions.

Usage

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

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

consensus_to_ppm(letter)

consensus_to_ppmAA(letter)

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

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

get_matches(motif, score)

get_scores(motif)

icm_to_ppm(position)

motif_score(motif, threshold = c(0, 1), use.freq = 1)

log_string_pval(pval)

pcm_to_ppm(position, pseudocount = 0)
```

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

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

ppm_to_pcm(position, nsites = 100)

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

pwm_to_ppm(position, bkg = numeric())

round_motif(motif, pct.tolerance = 0.05)

score_match(motif, match)

summarise_motifs(motifs, na.rm = TRUE)

ungap(motif, delete = FALSE)
```

Arguments

motif	Motif object to calculate scores from, or add/remove gap, or round.
gaploc	numeric Motif gap locations. The gap occurs immediately after every position value. If missing, uses round(ncol(motif) / 2).
mingap	numeric Minimum gap size. Must have one value for every location. If missing, set to 1.
maxgap	numeric Maximum gap size. Must have one value for every location. If missing, set to 5.
x	numeric First column for comparison.
y	numeric Second column for comparison.
method	character(1) Column comparison metric. See compare_motifs() for details.
bkg1	numeric Vector of background probabilities for the first column. Only relevant if method = "ALLR".
bkg2	numeric Vector of background probabilities for the second column. Only relevant if method = "ALLR".
nsites1	numeric(1) Number of sites for the first column. Only relevant if method = "ALLR".
nsites2	numeric(1) Number of sites for the second column. Only relevant if method = "ALLR".
letter	character(1) Any DNA, RNA, or AA IUPAC letter. Ambiguity letters are accepted.
position	numeric A numeric vector representing the frequency or probability for each alphabet letter at a specific position.
alphabet	character(1) One of c('DNA', 'RNA').
type	character(1) One of c('PCM', 'PPM', 'PWM' 'ICM').

pseudocount	numeric(1) Used to prevent zeroes in motif matrix.
score	numeric(1) Logodds motif score.
threshold	numeric(1) Any number of numeric values between 0 and 1 representing score percentage.
use.freq	numeric(1) Use regular motif or the respective multifreq representation.
pval	character(1) String-formatted p-value.
bkg	numeric Should be the same length as the alphabet length.
nsites	numeric(1) Number of sites motif originated from.
relative_entropy	logical(1) Calculate information content as relative entropy or Kullback-Leibler divergence.
schneider_correction	logical(1) Apply sample size correction.
smooth	logical(1) Apply pseudocount correction.
pct.tolerance	numeric(1) or character(1) The minimum tolerated proportion each letter must represent per position in order not to be rounded off, either as a numeric value from 0 to 1 or a percentage written as a string from "0%" to "100%".
match	character(1) Sequence string to calculate score from.
motifs	list A list of universalmotif motifs.
na.rm	logical Remove columns where all values are NA.
delete	logical(1) Clear gap information from motif. If FALSE, then it can be reactivated simply with add_gap(motif).

Value

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

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

For get_matches(): a character vector of motif matches.

For motif_score(): a named numeric vector of motif scores.

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

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

For ppm_to_icm(), icm_to_ppm(), pcm_to_ppm(), ppm_to_pcm(), ppm_to_pwm(), and pwm_to_ppm(): a numeric vector with length equal to input numeric vector.

For round_motif(): the input motif, rounded.

For score_match(): a numeric vector with the match motif score.

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

Author(s)

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

See Also

create_motif()

Examples

```
data(examplemotif)

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

##############################################################################
## compare_columns
## Compare two numeric vectors using the metrics from compare_motifs()
compare_columns(c(0.5, 0.1, 0.1, 0.2), c(0.7, 0.1, 0.1, 0.1), "PCC")

##############################################################################
## consensus_to_ppm
## Do the opposite of get_consensus. Note that loss of information is
## inevitable. Generates a sequence matrix.
sapply(c("A", "G", "T", "B"), consensus_to_ppm)

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

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

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

##############################################################################
## get_match
## Get all possible motif matches above input score
get_matches(examplemotif, 10)

##############################################################################
## get_scores
## Get all possible scores for a motif
length(get_scores(examplemotif))

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

```
#############################################################################
## motif_score
## Calculate motif score from different thresholds
m <- normalize(examplemotif)
motif_score(m, c(0, 0.8, 1))

#############################################################################
## log_string_pval
## Get the log of a string-formatted p-value
log_string_pval("1e-400")

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

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

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

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

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

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

#############################################################################
## 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)

#############################################################################
## round_motif
## Round down letter scores to 0
m <- create_motif()
```

```
## Remove letters from positions which are less than 5% of the total
## position:
round_motif(m, pct.tolerance = 0.05)

#############################################################################
## score_match
## Calculate score of a particular match
score_match(examplemotif, "TATATAT")
score_match(examplemotif, "TATATAG")

#############################################################################
## 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))

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

`utils-sequence` *Sequence-related utility functions.*

Description

Sequence-related utility functions.

Usage

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

get_klets(lets, k = 1)

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

Arguments

`string`	`character(1)` A length one character vector.
`k`	`integer(1)` K-let size.
`alph`	`character(1)` A single character string with the desired sequence alphabet. If missing, finds the unique letters in the string.
`lets`	`character` A character vector where each element will be considered a single unit.

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

Value

For count_klets(): A data.frame with columns lets and counts.

For get_klets(): A character vector of k-lets.

For shuffle_string(): A single character string.

Author(s)

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

See Also

create_sequences(), shuffle_sequences()

Examples

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

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

########################################################################
## shuffle_string
## Shuffle any string of characters
shuffle_string("ASDADASDASDASD", k = 2)
```

---

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 = "ALLR", tryRC = TRUE,
  min.overlap = 6, min.mean.ic = 0.25, relative_entropy = FALSE,
  normalise.scores = FALSE, min.position.ic = 0, score.strat = "sum",
  return.raw = FALSE, dedup.names = 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 PCC, EUCL, SW, KL, ALLR, BHAT, HELL, SEUCL, MAN, ALLR_LL, WEUCL, WPCC. See details.
tryRC	logical(1) Try the reverse complement of the motifs as well, report the best score.
min.overlap	numeric(1) Minimum overlap required when aligning the motifs. Setting this to a number higher then the width of the motifs will not allow any overhangs. Can also be a number between 0 and 1, representing the minimum fraction that the motifs must overlap.
min.mean.ic	numeric(1) Minimum mean information content between the two motifs for an alignment to be scored. This helps prevent scoring alignments between low information content regions of two motifs.
relative_entropy	logical(1) Change the ICM calculation affecting min.position.ic and min.mean.ic. See convert_type().
normalise.scores	logical(1) Favour alignments which leave fewer unaligned positions, as well as alignments between motifs of similar length. Similarity scores are multiplied by the ratio of aligned positions to the total number of positions in the larger motif, and the inverse for distance scores.
min.position.ic	numeric(1) Minimum information content required between individual alignment positions for it to be counted in the final alignment score. It is recommended to use this together with normalise.scores = TRUE, as this will help punish scores resulting from only a fraction of an alignment.
score.strat	character(1) How to handle column scores calculated from motif alignments. "sum": add up all scores. "a.mean": take the arithmetic mean. "g.mean": take the geometric mean. "median": take the median. "wa.mean", "wg.mean": weighted arithmetic/geometric mean. "fzt": Fisher Z-transform. Weights are the total information content shared between aligned columns.
return.raw	logical(1) Instead of returning a plot, return the aligned named matrices used to generate the plot. This can be useful if you wish to use view_motifs() alignment capabilities for custom plotting uses. Alignment is performed by adding empty columns to the left or right of motifs to generate matrices of equal length.
dedup.names	logical(1) Plotting motifs with duplicated names is not allowed. Setting this to TRUE allows the names to be modified for plotting.
...	Additional 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.

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

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

Value

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

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"][["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, overwrite = FALSE,
  append = FALSE)
```

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().
overwrite	logical(1) Overwrite existing file.
append	logical(1) Add to an existing file.

Value

NULL, invisibly.

Author(s)

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

References

Heinz S, Benner C, Spann N, Bertolino E, Lin YC, Laslo P, Cheng JX, Murre C, Singh H, Glass CK (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, overwrite = FALSE, append = FALSE)
```

Arguments

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

Value

NULL, invisibly.

Author(s)

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

References

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon JA, van der Lee R, Bessy A, Cheneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas DJ, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman WW, 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, overwrite = FALSE, append = FALSE)
```

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.
overwrite	logical(1) Overwrite existing file.
append	logical(1) Add to an existing file.

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 = 5, bkg, strand, overwrite = FALSE,
  append = FALSE)
```

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 "+ -"
overwrite	logical(1) Overwrite existing file.
append	logical(1) Add to an existing file. Motifs will be written in minimal format, so it is recommended to only use this if the existing file is also a minimal MEME format file.

Details

At this time non-DNA/RNA/AA alphabets are not supported.

Value

NULL, invisibly.

Author(s)

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

References

Bailey TL, Boden M, Buske FA, Frith M, Grant CE, Clementi L, Ren J, Li WW, Noble WS (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(). Currently the universalmotif format is YAML-based, but this is subject to change.

Usage

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

Arguments

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.
progress	logical(1) Show progress.
overwrite	logical(1) Overwrite existing file.
append	logical(1) Add to an existing motif file. Package version in existing motif file must be greater than 1.2.0.
BP	logical(1) Allows for the use of **BiocParallel** within write_motifs(). See BiocParallel::register() to change the default backend.

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, overwrite = FALSE, append = FALSE)
```

Arguments

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

Value

NULL, invisibly.

Author(s)

Benjamin Jean-Marie Tremblay, <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