# Package 'memes'

June 28, 2022

**Type** Package

**Title** motif matching, comparison, and de novo discovery using the MEME Suite

**Version** 1.4.1

**Description** A seamless interface to the MEME Suite family of tools for motif analysis. 'memes' provides data aware utilities for using GRanges objects as entrypoints to motif analysis, data structures for examining & editing motif lists, and novel data visualizations. 'memes' functions and data structures are amenable to both base R and tidyverse workflows.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Imports** Biostrings, dplyr, cmdfun (>= 1.0.2), GenomicRanges, ggplot2, ggseqlogo, magrittr, matrixStats, methods, patchwork, processx, purrr, rlang, readr, stats, tools, tibble, tidyr, utils, usethis, universalmotif (>= 1.9.3), xml2

**Suggests** cowplot, BSgenome.Dmelanogaster.UCSC.dm3, BSgenome.Dmelanogaster.UCSC.dm6, forcats, testthat (>= 2.1.0), knitr, MotifDb, pheatmap, PMCMRplus, plyranges (>= 1.9.1), rmarkdown, covr

**biocViews** DataImport, FunctionalGenomics, GeneRegulation, MotifAnnotation, MotifDiscovery, SequenceMatching, Software

**RoxygenNote** 7.1.1

**SystemRequirements** Meme Suite (v5.3.3 or above) <http://meme-suite.org/doc/download.html>

**VignetteBuilder** knitr

**URL** https://snystrom.github.io/memes/, https://github.com/snystrom/memes

**BugReports** https://github.com/snystrom/memes/issues

**Depends** R (>= 4.1)

**Config/testthat/edition** 3

**git_url** https://git.bioconductor.org/packages/memes

**git_branch** RELEASE_3_15

**git_last_commit** affa506

**git_last_commit_date** 2022-05-08

**Date/Publication** 2022-06-28

**Author** Spencer Nystrom [aut, cre, cph]
(<https://orcid.org/0000-0003-1000-1579>)

**Maintainer** Spencer Nystrom <nystromdev@gmail.com>

# R topics documented:

---

add_sequence *Add nucleic acid sequence of regions to metadata column*

---

## Description

Add nucleic acid sequence of regions to metadata column

## Usage

```
add_sequence(ranges, genome, name = "sequence")
```

## Arguments

| | |
|---|---|
| ranges | GRanges object |
| genome | BSgenome object or any other valid input to 'Biostrings::getSeq()' (Do 'showMethods(Biostrings::getSeq)' to show valid types) |
| name | name of metadata column to hold sequence information (default: "sequence"). Note, this will overwrite existing columns without warning if the name already exists. |

## Value

'ranges' with new metadata column named "sequence" (or another value passed to 'name') holding the DNA or RNA sequence from 'genome'

## Examples

```
data(example_peaks, package = "memes")
dm.genome <- BSgenome.Dmelanogaster.UCSC.dm3::BSgenome.Dmelanogaster.UCSC.dm3
add_sequence(example_peaks, dm.genome)
```

## ame_compare_heatmap_methods *Compare AME heatmap methods*

### Description

This helper function allows the user to visualize the distribution of their AME results data on different scales to help understand the implications of using different values in ‘plot_ame_heatmap()‘

### Usage

```
ame_compare_heatmap_methods(ame, group, value = -log10(adj.pvalue))
```

### Arguments

| | |
|---|---|
| ame | ame results data.frame |
| group | optional name of group to split results by |
| value | value to compare to "normalize" method (default: -log10(adj.pvalue)) |

### Value

a cowplot 2 panel plot comparing the distribution of ‘value‘ to normalized rank values

### Examples

```
data("example_ame", package = "memes")
ame_compare_heatmap_methods(example_ame$Decreasing)

ame_compare_heatmap_methods(dplyr::bind_rows(example_ame, .id = "type"), type)
```

## check_meme_install *Check user's MEME install*

### Description

In order to use the run* family of functions, memes must detect a local install of the MEME Suite. MEME is installed in a directory named meme/bin/ which can be located anywhere on the filesystem, but is typically found in `~/meme/bin`. If the MEME Suite is installed at `~/meme/bin`, memes can autodetect the install. However, in the case that the MEME Suite is found at a nonstandard location, the user may specify the location of their meme/bin in three ways:

### Usage

```
check_meme_install(meme_path = NULL)
```

### Arguments

| | |
|---|---|
| meme_path | path to "meme/bin" (if unset will search MEME_BIN environment variable or meme_bin option) |

### Details

1. provide the full path to meme/bin to the meme_path argument to each run* function.
2. set the meme_bin option using options(meme_bin = "path/to/meme/bin") once per R session.
3. set the MEME_BIN environment variable either in .Renviron or ~/.bashrc with the path to meme/bin

To aid the user in determining if memes can detect their meme/bin install, check_meme_install() will search the aforementioned locations for a valid meme/bin, returning green checks for each detected tool, or red X's for undetected tools. Alternatively, users can run meme_is_installed() to get a boolean value indicating whether their MEME Suite can be detected.

check_meme_install() searches using the following heirarchy. This heirarchy mimics how all run* functions search for meme/bin, thus the paths printed from check_meme_install() will indicate the paths used by each run* function. The heirarchy is as follows:

1. the meme_path function argument if set
2. the meme_bin option
3. the MEME_BIN environment variable
4. the default location at ~/meme/bin

### Value

message indicating which MEME utilities are installed and their location on disk

### Examples

```
check_meme_install()
```

---

## drop_best_match *Drop best match columns from tomtom results*

---

### Description

Convenience function for dropping all columns created by runTomTom prefixed by "best_match_" and the "best_db_name" column. Keeps the "tomtom" data.frame column. Can be useful if you want to unnest the 'tomtom' data without propagating these columns.

### Usage

```
drop_best_match(res)
```

### Arguments

| | |
|---|---|
| res | results of runTomTom |

### Value

'res' without the tomtom best_match_ columns

### Examples

```
data("example_dreme_tomtom")
names(example_dreme_tomtom)
names(drop_best_match(example_dreme_tomtom))
```

---

## example_ame *Example runAme() output*

---

### Description

Result when running AME using 100bp window around 'example_chip_summits' for "Increasing" and "Decreasing" sites, using "Static" as background.

### Usage

```
example_ame
```

### Format

A list object of AME results data.frames

**Increasing** 'runAme()' Results object for Increasing sites vs Static sites

**Decreasing** 'runAme()' Results object for Decreasing sites vs Static sites

### Examples

```
# Data can be combined into 1 large data.frame using:
# where the "behavior" column will hold the "Increasing"/"Decreasing" information
dplyr::bind_rows(example_ame, .id = "behavior")
```

## example_ame_large *runAme() output for example_chip_summits split by binding description*

### Description

AME was run for "ectopic", "entopic", and "orphan" sites using shuffled background.

### Usage

```
example_ame_large
```

### Format

a list of runAme() results data.frames

### Details

see ‘vignette("integrative_analysis", package = "memes")‘ for details.

### Examples

```
# Data can be combined into 1 large data.frame using:
dplyr::bind_rows(example_ame_large, .id = "binding_type")
```

## example_chip_summits *Annotated Transcription Factor ChIP-seq summits*

### Description

ChIP-seq summit positions on Drosophila melanogaster chromosome 3 for the transcription factor E93 using a union set of peaks from third-instar larval wings ("Early") and 24 hour Pupal ("Late") wings.

### Usage

```
example_chip_summits
```

### Format

A GRanges object of ChIP summit position with 2 metadata columns

**peak_binding_description** Binding profiles between Early and Late E93 were compared. Peaks are annotated as whether they are bound in Early wings only ("ectopic"), both Early and Late wings ("entopic"), or only bound in Late wings ("orphan").

**e93_sensitive_behavior** change in chromatin accessibility in response to E93 binding: Increasing, Decreasing, or Static

## Details

E93 is a transcription factor normally present only in Late wings. An experimental perturbation precociously expressed E93 during Early stages. Binding profiles between Early and Late E93 were compared. Peaks are annotated as whether they are bound in Early wings only ("ectopic"), both Early and Late wings ("entopic"), or only bound in Late wings ("orphan").

DNA elements can be made "open" or "closed" in response to binding of transcription factors like E93. Accessibility of E93 binding sites before and after E93 expression was measured using FAIRE-seq. ChIP peaks are annotated by how their accessibility changes in response to E93 binding . Peaks can become more open ("Increasing"), more closed ("Decreasing"), or unchanged in accessibility ("Static"). These experiments demonstrate a causal relationship between E93 binding and both opening and closing of DNA elements.

## Source

https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE141738&format=file&file=GSE141738

---

`example_dreme` *Example runDreme() output*

---

## Description

Result when running dreme using 100bp window around example_chip_summits using "Decreasing" sites as foreground, and "Static" sites as background.

## Usage

```
example_dreme
```

## Format

a runDreme results data.frame

---

`example_dreme_by_binding` *runDreme() output for example_chip_summits split by binding description*

---

## Description

See vignette("integrative_analysis", package = "memes") for details

## Usage

```
example_dreme_by_binding
```

## Format

a runDreme results data.frame

## example_dreme_tomtom *Example runDreme() output after passing to runTomTom()*

### Description

Result when running ‘runTomTom(example_dreme)‘ using FlyFactorSurvey as database.

### Usage

```
example_dreme_tomtom
```

### Format

a runDreme results data.frame with runTomTom results columns

## example_fimo *Example runFimo() output*

### Description

Run using 100bp windows around ‘example_chip_summits‘, using E93 motif as database.

### Usage

```
example_fimo
```

### Format

A GRanges object of E93 motif positions within 100bp windows of ‘example_chip_summits‘

## example_peaks *Example ChIP-seq peaks*

### Description

10 ChIP-seq peaks from GSE141738

### Usage

```
example_peaks
```

### Format

An object of class `GRanges` of length 10.

## Details

A small number of transcription factor ChIP-seq peaks as a GRanges object, taken from [GSE141738](https://www.ncbi.nlm.n

## Source

https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE141738&format=file&file=GSE141738

---

`example_rnaseq` *RNAseq data from Early and Late Drosophila wings*

---

## Description

These data are a subset of RNAseq counts from the full FPKM table in GSE97956. Includes counts for all Drosophila transcription factors and ~200 other random genes.

## Usage

```
example_rnaseq
```

## Format

A data.frame of RNAseq FPKM data

**symbol** The FlyBase gene symbol

**time** Developmental stage of RNAseq experiment

**fpkm** RNAseq count in Fragments per Kilobase Million (FPKM)

## Source

"https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE97956&format=file&file=GSE97956

---

`example_tomtom` *Example runTomTom() output*

---

## Description

Result when running ‘runTomTom(example_dreme$motif)‘ using FlyFactorSurvey as database

## Usage

```
example_tomtom
```

## Format

a data.frame

`force_best_match` *Force best tomtom match by id*

## Description

Although TomTom assigns a best match, this is not always the most biologically relevant match. In these cases, it is useful to "force" the best match to another lower ranked, but still significant TomTom match. This function allows users to select a new best match motif from the set of lower-ranked matches in the ‘tomtom‘ list column. This function also reorders the ‘tomtom‘ data.frame such that the forced match is the first row of the ‘tomtom‘ entry.

## Usage

```
force_best_match(res, matches)
```

## Arguments

| | |
|---|---|
| `res` | results from runTomTom |
| `matches` | named vector where name is the input motif name, and value is the match_name to use as the new best match |

## Value

‘res‘ with new best_* columns and re-ranked tomtom data in the ‘tomtom‘ list column for the updated entries.

## See Also

[update_best_match()]

## Examples

```
if (meme_is_installed()){
motif <- universalmotif::create_motif("CCRAAAW", name = "example_motif")
db <- system.file("extdata", "flyFactorSurvey_cleaned.meme", package = "memes")
res <- runTomTom(motif, database = db)
res$best_match_name
res2 <- force_best_match(res, c("example_motif" = "Eip93F_SANGER_10"))
res2$best_match_name
}
```

get_sequence *Get sequence from GRanges*

## Description

A light wrapper around Biostrings::getSeq to return named DNAStringSets, from input genomic coordinates.

## Usage

```
get_sequence(regions, genome, score_column, ...)
```

## Arguments

| | |
|---|---|
| regions | GRanges, or GRangesList object. Will also accept a data.frame as long as it can be coerced to a GRanges object, or a string in the format: "chr:start-end" (NOTE: use 1-based closed intervals, not BED format 0-based half-open intervals). |
| genome | object of any valid type in 'showMethods(Biostrings::getSeq)'. Commonly a BSgenome object, or fasta file. Used to look up sequences in regions. |
| score_column | optional name of column (in mcols() of 'regions') containing a fasta score that is added to the fasta header of each entry. Used when using [runAme()] in partitioning mode. (default: 'NULL') |
| ... | additional arguments passed to Biostrings::getSeq. |

## Value

'Biostrings::DNAStringSet' object with names corresponding to genomic coordinates. If input is a list object, output will be a 'Biostrings::BStringSetList' with list names corresponding to input list names.

## Examples

```
# using character string as coordinates
# using BSgenome object for genome
drosophila.genome <- BSgenome.Dmelanogaster.UCSC.dm6::BSgenome.Dmelanogaster.UCSC.dm6
get_sequence("chr2L:100-200", drosophila.genome)

# using GRanges object for coordinates
data(example_peaks, package = "memes")
get_sequence(example_peaks, drosophila.genome)
```

## has_duplicate_motifs *Check for duplicated motif matrices*

### Description

This function identifies whether any motif matrices in the input universalmotif list or universalmotif_df are identical to each other. Note: this operation is slow on large motif lists

### Usage

```
has_duplicate_motifs(x)
```

### Arguments

| | |
|---|---|
| x | a universalmotif list or universalmotif_df |

### Value

logical value indicating presence or absence of duplicated motif matrices

### Examples

```
motif <- universalmotif::create_motif()
duplicated <- c(motif, motif)
has_duplicate_motifs(duplicated)
```

## importAme *Parse AME output*

### Description

This imports AME results using the "ame.tsv" output, and optionally the "sequences.tsv" output if run with "method = fisher". AME results differ based on the method used, thus this must be set on import or the column names will be incorrect.

### Usage

```
importAme(path, method = "fisher", sequences = FALSE)
```

### Arguments

| | |
|---|---|
| path | path to ame results file ("ame.tsv") |
| method | ame run method used (one of: c("fisher", "ranksum", "dmhg3", "dmhg4", "pearson", "spearman")). Default: "fisher". |
| sequences | FALSE or path to sequences file (only valid for method = "fisher") |

## Value

data.frame with method-specific results. See [AME results](http://meme-suite.org/doc/ame-output-format.html) webpage for more information. If sequences is set to a path to the sequences.tsv and method = "fisher", the list-column ‘sequences‘ will be appended to resulting data.frame.

## See Also

[runAme()]

## Examples

```
ame_tsv <- system.file("extdata", "ame.tsv", package = "memes", mustWork = TRUE)
importAme(ame_tsv)
```

---

`importDremeXML` *Import Dreme output from previous run*

---

## Description

Import Dreme output from previous run

## Usage

```
importDremeXML(dreme_xml_path)
```

## Arguments

| | |
|---|---|
| `dreme_xml_path` | path to dreme.xml file |

## Value

data.frame with statistics for each discovered motif. The ‘motifs‘ column contains a universalmotif object representation in PCM format of each DREME motif. If no motifs are discovered, returns NULL.

## See Also

[runDreme()]

## Examples

```
dreme_xml <- system.file("extdata", "dreme.xml", package = "memes")
importDremeXML(dreme_xml)
```

importFimo *Import FIMO results*

## Description

Import FIMO results

## Usage

```
importFimo(fimo_tsv)
```

## Arguments

| | |
|---|---|
| fimo_tsv | path to fimo.tsv output file |

## Value

GenomicRanges object for each match position. Note unless coordinates are genomic positions, each ‘seqnames‘ entry will be the fasta header, and start/end will be the position within that sequence of the match.

## Examples

```
fimo_tsv <- system.file("extdata", "fimo.tsv", package = "memes")
importFimo(fimo_tsv)
```

importMeme *Import MEME results*

## Description

This is a light wrapper around [universalmotif::read_meme()] that imports MEME results as universalmotif data.frame. If MEME is run with genomic coordinates in the fasta header, in "chr:start-end" format (base 1 indexed), the genomic coordinates of the motif match from input sequences can be parsed from the header.

## Usage

```
importMeme(meme_txt, parse_genomic_coord = FALSE, combined_sites = FALSE)
```

### Arguments

| | |
|---|---|
| meme_txt | path to "meme.txt" output |
| parse_genomic_coord | whether to parse sequence headers into genomic coordinates for motif position information, only works if fasta files were written such that the sequence headers are in the form: "chr:start-end", or some variation of this form (delimiters can be any of: "[^[:alnum:]]+" (ie non-alphanumeric characters)), (default = FALSE). |
| combined_sites | whether to add 'combined_sites' output which contains coordinates of each sequence, the motif sequence (if 'parse_genomic_coord = TRUE'), and the 'diagram' column raw output from MEME indicating the relative locations of motifs in the sequence. |

### Value

MEME results in universalmotif data.frame format (see: [as_universalmotif_dataframe()]). 'sites_hits' is a nested data.frame column containing the position within each input sequence of matches to the identified motif.

### See Also

[runMeme()] [universalmotif::read_meme()]

### Examples

```
example_meme_txt <- system.file("extdata", "meme_full.txt", package = "universalmotif")
importMeme(example_meme_txt)
```

---

## importStremeXML *Import Streme output from previous run*

---

### Description

Import Streme output from previous run

### Usage

```
importStremeXML(streme_xml_path)
```

### Arguments

| | |
|---|---|
| streme_xml_path | path to streme.xml file |

### Value

data.frame with statistics for each discovered motif. The 'motifs' column contains a universalmotif object representation in PCM format of each DREME motif. If no motifs are discovered, returns NULL.

## See Also

[runStreme()]

## Examples

```
streme_xml <- system.file("extdata", "streme.xml", package = "memes")
importStremeXML(streme_xml)
```

---

`importTomTomXML` *Import tomtom data from previous run*

---

## Description

Import tomtom data from previous run

## Usage

```
importTomTomXML(tomtom_xml_path)
```

## Arguments

`tomtom_xml_path`
: path to tomtom.xml

## Details

tomtom list column format the 'tomtom' list column contains data.frames with the following format: - name: name of query PWM - altname: alternate name of query PWM - match_name: name of matched PWM - match_altname: alt name of matched PWM - match_pval: p-value of match - match_eval: E-value of match - match_qval: q-value of match - match_offset: number of letters the query was offset from the target match - match_strand: whether the motif was found on input strand (+) or as reverse-complement (-) - db_name: database source of matched motif - match_motif: universalmotif object containing the PWM that was matched

## Value

will return data.frame with input motifs & results for best match. 'tomtom' list column contains full tomtom data for each input motif. NOTE: if tomtom detects no matches for any input motif, currently will print a message & return NA values for 'tomtom', 'best_match_name', and 'best_match_motif'.

## See Also

[runTomTom()]

## Examples

```
tomtom_xml <- system.file("extdata", "tomtom.xml", package = "memes")
importTomTomXML(tomtom_xml)
```

`meme_is_installed` *Returns logical vector indicating valid MEME-Suite install status*

## Description

Checks for a valid meme install using same heirarchy as `check_meme_install()`. Returns `TRUE` if all supported utilities are found in the meme install location, `FALSE` if not.

## Usage

```
meme_is_installed(path = NULL)
```

## Arguments

| | |
|---|---|
| `path` | optional path to "meme/bin/". If unset, will follow the search heirarchy listed above. |

## Details

The search heirarchy is as follows:

1. the `meme_path` function argument if set
2. the `meme_bin` option
3. the `MEME_BIN` environment variable
4. the default location at `~/meme/bin`

## Value

`logical(1)` indicating whether meme is installed with all supported utilities

## See Also

`check_meme_install()`

## Examples

```
meme_is_installed()
```

`nest_tomtom` *Nest TomTom results columns into a data.frame column named "tomtom"*

## Description

This is a convienience function for re-nesting the ‘tomtom‘ list column if the user unnests it. Additionally, it will update the best_match information based on the ranking of the resulting ‘tomtom‘ data.frame. This avoids having out-of-date best_match information after manipulating the ‘tomtom‘ entries.

## Usage

```
nest_tomtom(data)
```

## Arguments

| | |
|---|---|
| `data` | tomtom results data.frame after unnesting the ‘tomtom‘ column |

## Details

**NOTE:** that the resulting columns may not be in the same order, so operations like ‘identical()‘ before & after a nest/renest operation may fail even though the column values are unchanged.

## Value

the input data.frame with the match_* columns nested into a column named ‘tomtom‘

## Examples

```
if (meme_is_installed()){
motif <- universalmotif::create_motif("CCRAAAW")
db <- system.file("extdata/flyFactorSurvey_cleaned.meme", package = "memes")
res <- runTomTom(motif, database = db)
data <- tidyr::unnest(res, "tomtom")
identical(nest_tomtom(data), res)
}
```

`plot_ame_heatmap` *Plot AME heatmap clustered by similarity in detected motifs*

## Description

Plot AME heatmap clustered by similarity in detected motifs

## Usage

```
plot_ame_heatmap(
  ame,
  id = motif_id,
  group = NULL,
  value = -log10(adj.pvalue),
  group_name = NULL,
  scale_max = NA
)
```

## Arguments

| | |
|---|---|
| ame | ame results data.frame |
| id | column of motif ids to use (default: motif_id). |
| group | grouping column if comparing across multiple ame runs (optional, default: NULL). |
| value | value to display as heatmap intensity. Default: -log10(adj.pvalue). Takes function or column name as input. If set to "normalize", will use normalized rank within ‘group‘ as the heatmap values. **If in doubt**, prefer the -log10(adj.pvalue) plot potentially in conjunction with adjusting ‘scale_max‘. (See "Normalized rank visualization" section below for more details on how to interpret these data) |
| group_name | when group = NULL, name to use for input regions. Ignored if group is set. |
| scale_max | max heatmap value to limit upper-value of scale. Useful if distribution of ‘value‘s vary greatly between groups. Usually a better to tweak this option than to use value = "normalize". The cumulative distribution plot generated by ‘ame_compare_heatmap_methods()‘ can be useful for selecting this value, try to pick a value which captures the largest fraction of hits across all groups while excluding outliers. |

## Details

Normalized rank visualization **NOTE:** The normalized rank visualization eliminates all real values related to statistical significance! Instead, this visualization represents the relative ranks of hits within an AME run, which already pass a significance threshold set during ‘runAME()‘. This means that even if several motifs have similar or even identical pvalues, their heatmap representation will be a different color value based on their ranked order in the results list. This also means that using the normalized rank visualization will be misleading if there are only a few AME hits; it is only worth using if the number of hits is very large (>100). Both visualizations can be useful and reveal different properties of the data to the user during data exploration. Use ‘ame_compare_heatmap_methods()‘ to help assess differences in the two visualizations. **If in doubt**, prefer the ‘-log10(adj.pvalue)‘ representation.

Common mistake: if ‘value‘ is set to a string that is not "normalize", it will return: "Error: Discrete value supplied to continuous scale". To use a column by name, do not quote the column name.

## Value

‘ggplot‘ object

## Examples

```
data("example_ame", package = "memes")

# Plot a single category heatmap
plot_ame_heatmap(example_ame$Decreasing)

# Plot a multi category heatmap
grouped_ame <- dplyr::bind_rows(example_ame, .id = "category")
plot_ame_heatmap(grouped_ame, group = category)
```

---

# plot_sequence_heatmap *Visualize a heatmap of aligned sequences*

---

## Description

Sometimes it is useful to visualize individual motif matches in aggregate to understand how sequence variability contributes to motif matches. This function creates a heatmap where each row represents a single sequence and each column represents a position. Cells are colored by the sequence at that position. Sequences are optionally aggregated into a sequence logo aligned in register with the heatmap to visualize how sequence variability contributes to motif makeup.

## Usage

```
plot_sequence_heatmap(
  sequence,
  title = NULL,
  logo = TRUE,
  alph = c("DNA", "RNA", "AA"),
  title_hjust = 0,
  heights = c(1, 5),
  legend = "none"
)
```

## Arguments

| | |
|---|---|
| sequence | character vector of sequences, plot will be ranked in order of the sequences. Each sequence must be equal length. Alternately, sequence can be a named list in which case each plot will be titled by the names of the list entries. |
| title | title of the plot. Default: NULL. If sequence is a named list of sequences, title defaults to the list entry names. Set to NULL to override this behavior. To use a different title than the list entry name, pass a vector of names to 'title'. |
| logo | whether to include a sequence logo above the heatmap |
| alph | alphabet colorscheme to use. One of: DNA, RNA, AA. |
| title_hjust | value from 0 to 1 determining the horizontal justification of the title. Default: 0. |
| heights | ratio of logo:heatmap heights. Given as: c(logo_height, heatmap_height). Values are not absolute. Ignored when logo = FALSE. |
| legend | passed to ggplot2::theme(legend.position). Default: "none". Values can be: "none", "left", "right", "top", "bottom", or coordinates in c(x,y) format. |

## Value

a ggplot object of the sequence heatmap ranked by the order of sequences

## See Also

runFimo

## Examples

```
data(example_fimo, package = "memes")
genome <- BSgenome.Dmelanogaster.UCSC.dm3::BSgenome.Dmelanogaster.UCSC.dm3
motifs <- add_sequence(example_fimo, genome)
plot_sequence_heatmap(motifs$sequence)

# Use on named list
sequences <- list("set 1" = motifs$sequence[1:100],
                  "set 2" = motifs$sequence[101:200])
plot_sequence_heatmap(sequences)

# Use different titles for list input
plot_sequence_heatmap(sequences, title = c("A", "B"))
```

---

# remove_duplicate_motifs

*Remove duplicated motif entries*

---

## Description

This function identifies motif matrices which are duplicated in a universalmotif list or universalmotif_df and removes them. This operation ignores motif metadata and operates by removing all entries whose motif matrices are identical. The first instance of a duplicated motif in the input list is the one returned.

## Usage

```
remove_duplicate_motifs(x)
```

## Arguments

| | |
|---|---|
| x | a universalmotif list or universalmotif_df |

## Value

A deduplicated list or universalmotif_df

## Examples

```
motif <- universalmotif::create_motif()
duplicated <- c(motif, motif)
remove_duplicate_motifs(duplicated)
```

runAme.list *Motif enrichment using AME*

## Description

AME identifies known motifs (provided by the user) that are enriched in your input sequences.

## Usage

```
## S3 method for class 'list'
runAme(
  input,
  control = "shuffle",
  outdir = "auto",
  method = "fisher",
  database = NULL,
  meme_path = NULL,
  sequences = FALSE,
  silent = TRUE,
  ...
)

## S3 method for class 'BStringSetList'
runAme(
  input,
  control = "shuffle",
  outdir = "auto",
  method = "fisher",
  database = NULL,
  meme_path = NULL,
  sequences = FALSE,
  silent = TRUE,
  ...
)

## Default S3 method:
runAme(
  input,
  control = "shuffle",
  outdir = "auto",
  method = "fisher",
  database = NULL,
  meme_path = NULL,
  sequences = FALSE,
  silent = TRUE,
  ...
)
```

```
runAme(
  input,
  control = "shuffle",
  outdir = "auto",
  method = "fisher",
  database = NULL,
  meme_path = NULL,
  sequences = FALSE,
  silent = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| input | path to fasta, or DNAstringset (optional: DNAStringSet object names contain fasta score, required for partitioning mode) |
| control | default: "shuffle", or set to DNAstringset or path to fasta file to use those sequences in discriminative mode. If input is a list of DNAStringSet objects, and control is set to a character vector of names in input, those regions will be used as background in discriminitive mode and AME will skip running on any control entries (NOTE: if input contains an entry named "shuffle" and control is set to "shuffle", it will use the input entry, not the AME shuffle algorithm). If control is a Biostrings::BStringSetList (generated using get_sequence), all sequences in the list will be combined as the control set. Set to NA for partitioning based on input fasta score (see get_sequence() for assigning fasta score). If input sequences are not assigned fasta scores but AME is run in partitioning mode, the sequence order will be used as the score. |
| outdir | Path to output directory location to save data files. If set to "auto", will use location of input files if passing file paths, otherwise will write to a temporary directory. default: "auto" |
| method | default: fisher (allowed values: fisher, ranksum, pearson, spearman, 3dmhg, 4dmhg) |
| database | path to .meme format file, universalmotif list object, dreme results data.frame, or list() of multiple of these. If objects are assigned names in the list, that name will be used as the database id in the output. It is highly recommended you set a name if not using a file path so the database name will be informative, otherwise the position in the list will be used as the database id. For example, if the input is: list("motifs.meme", list_of_motifs), the database id's will be: "motifs.meme" and "2". If the input is list("motifs.meme", "customMotifs" = list_of_motifs), the database id's will be "motifs.meme" and "customMotifs". |
| meme_path | path to "meme/bin/" (default: NULL). Will use default search behavior as described in check_meme_install() if unset. |
| sequences | logical(1) add results from sequences.tsv to sequences list column to returned data.frame. Valid only if method = "fisher". See AME outputs webpage for more information (Default: FALSE). |
| silent | whether to suppress stdout (default: TRUE), useful for debugging. |

... additional arguments passed to AME (see AME Flag table below)

## Details

AME can be run using several modes:

1. Discriminative mode: to discover motifs enriched relative to shuffled input, or a set of provided control sequences
2. Partitioning mode: in which AME uses some biological signal to identify the association between the signal and motif presence.

To read more about how AME works, see the AME Tutorial

Additional AME arguments

memes allows passing any valid flag to it's target programs via .... For additional details for all valid AME arguments, see the AME Manual webpage. For convenience, a table of valid parameters, and brief descriptions of their function are provided below:

| AME Flag | allowed values | default | description |
|---|---|---|---|
| kmer | integer(1) | 2 | kmer frequency to preserve when shufflin |
| seed | integer(1) | 1 | seed for random number generator when |
| scoring | "avg", "max", "sum", "totalhits" | "avg" | Method for scoring a sequence for match |
| hit_lo_fraction | numeric(1) | 0.25 | fraction of hit log odds score to exceed to |
| evalue_report_threshold | numeric(1) | 10 | E-value threshold for reporting a motif as |
| fasta_threshold | numeric(1) | 0.001 | AME will classify sequences with FASTA |
| fix_partition | numeric(1) | NULL | AME evaluates only the partition of the f |
| poslist | "pwm", "fasta" | "fasta" | When using paritioning mode (control = |
| log_fscores | logical(1) | FALSE | Convert FASTA scores into log-space (or |
| log_pwmscores | logical(1) | FALSE | Convert PWM scores into log-space (onl |
| lingreg_switchxy | logical(1) | FALSE | Make the x-points FASTA scores and y-p |
| xalph | file path | NULL(1) | alphabet file to use if input motifs are in |
| bfile | "motif", "motif-file", "uniform", path to file | NULL | source of 0-order background model. If " |
| motif_pseudo | numeric(1) | 0.1 | Addd this pseudocount when converting |
| inc | character(1) | NULL | use only motifs with names matching thi |
| exc | character(1) | NULL | exclude motifs with names matching this |

## Value

a data.frame of AME enrichment results. If run using a BStringsSetList input, returns a list of data.frames.

## Citation

If you use runAme() in your analysis, please cite:

Robert McLeay and Timothy L. Bailey, "Motif Enrichment Analysis: A unified framework and method evaluation", BMC Bioinformatics, 11:165, 2010, doi:10.1186/1471-2105-11-165. full text

**Licensing:**

The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the MEME Suite Copyright Page for details.

## Examples

```
if (meme_is_installed()) {
# Create random named sequences as input for example
seqs <- universalmotif::create_sequences(rng.seed = 123)
names(seqs) <- seq_along(seqs)

# An example path to a motif database file in .meme format
motif_file <- system.file("extdata", "flyFactorSurvey_cleaned.meme", package = "memes")

runAme(seqs, database = motif_file)

# Dreme results dataset for example
dreme_xml <- system.file("extdata", "dreme.xml", package = "memes")
dreme_results <- importDremeXML(dreme_xml)

# database can be set to multiple values like so:
runAme(seqs, database = list(motif_file, "my_dreme_motifs" = dreme_results))
}
```

---

# runDreme *Denovo motif discovery of target regions using DREME*

---

## Description

DREME discovers short, ungapped, *de-novo* motifs that are relatively enriched relative to a control set of sequences. DREME can be run to discover motifs relative to a shuffled set of input sequences, or against a separately provided set of "control" sequences.

## Usage

```
runDreme(input, control, outdir = "auto", meme_path = NULL, silent = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `input` | regions to scan for motifs. Can be any of:<br>• path to fasta file<br>• DNAStringSet object (can be generated from GRanges using `get_sequence()`)<br>• List of DNAStringSet objects (generated from `get_sequence()`)<br>• *NOTE:* if using StringSet inputs, each entry must be named (set with `names()`).<br>• *NOTE:* If you want to retain the raw dreme output files, you must use a path to fasta file as input, or specify an "outdir" |
| `control` | regions to use as background for motif search. Can be any of:<br>• path to fasta file |

- DNAStringSet object (can be generated from GRanges using get_sequence)
- A Biostrings::BStringSetList (generated using `get_sequence`), in which case all sequences in the list will be combined as the control set.
- if `input` is a list of DNAStringSet objects, a character vector of names in `input` will use those sequences as background. runDreme will not scan those regions as input.
- "shuffle" to use dreme's built-in dinucleotide shuffle feature (NOTE: if `input` is a list object with an entry named "shuffle", the list entry will be used instead). Optionally can also pass `seed = <any number>` to `...` to use as the random seed during shuffling. If no seed is passed, dreme will use 1 as the random seed, so results will be reproducible if rerunning. **NOTE:** beware system-specific differences. As of v5, dreme will compile using the default python installation on a system (either python2.7 or python3). The random number generator changed between python2.7 and python3, so results will not be reproducible between systems using python2.7 vs 3.

`outdir` path to output directory of dreme files, or "auto" to autogenerate path. Default: location of input fasta in dir named "\<input\>*vs*\<control\>". If input is DNAstringset, will be temporary path. This means that if you want to save the raw output files, you must use fasta files as input or use an informative (and unique) outdir name. memes will **not check** if it overwrites files in a directory. Directories will be recursively created if needed.

`meme_path` optional, path to "meme/bin/" on your local machine. runDreme will search 3 places in order for meme if this flag is unset:

1. the option "meme_bin" (set with options(meme_bin = "path/to/meme/bin"))
2. the environment variable "MEME_PATH" (set in .Renviron)
3. "~/meme/bin/" as the default location

- If the user sets meme_path in the function call, this value will always be used

`silent` whether to suppress printing dreme stdout as a message when finishing with no errors. Can be useful for troubleshooting in situations where no motifs are discovered, but command completes successfully. (default: TRUE)

`...` dreme flags can be passed as R function arguments to use non-default behavior. For a full list of valid arguments, run your local install of dreme -h, or visit the dreme documentation website. See list below for aliases of common flags. To set flags with no values (ex. `-dna`), pass the argument as a boolean value (ex. `dna = TRUE`).

## Details

Properly setting the `control` parameter is key to discovering biologically relevant motifs. Often, using `control = "shuffle"` will produce a suboptimal set of motifs; however, some discriminative analysis designs don't have proper "control" regions other than to shuffle.

As of MEME version 5.2.0, DREME is deprecated. Consider `runStreme()` instead.

In addition to allowing any valid flag of dreme to be passed to `...`, we provide a few user-friendly aliases for common flags which are more readable (see list below). For example, e = 1 will use a

max evalue cutoff of 1. This is equivalent to setting evalue = 1. For additional details about each DREME flag, see the DREME Manual Webpage.

List of values which can be passed to ...: **NOTE:** values must be referred to using their name in the "memes alias" column, not the "DREME Flag" column.

| memes alias | DREME Flag | description | default |
|---|---|---|---|
| nmotifs | m | max number of motifs to discover | NULL |
| sec | t | max number of seconds to run | NULL |
| evalue | e | max E-value cutoff | 0.05 |
| seed | s | random seed if using "shuffle" as control | 1 |
| ngen | g | nuber of REs to generalize | 100 |
| mink | mink | minimum motif width to search | 3 |
| maxk | maxk | maximum motif width to search | 7 |
| k | k | set mink and maxk to this value | NULL |
| norc | norc | search only the input strand for sequences | FALSE |
| dna | dna | use DNA alphabet | TRUE |
| rna | rna | use RNA alphabet | FALSE |
| protein | protein | use protein alphabet (NOT RECCOMENDED) | FALSE |

## Value

`universalmotif_df` with statistics for each discovered motif. The `motif` column contains a universalmotif object representation in PCM format of each DREME motif. If no motifs are discovered, returns NULL. The column values are as follows:

- rank = ranked order of discovered motif
- name = primary name of motif
- altname = alternative name of motif
- seq = consensus sequence of the motif
- length = length of discovered motif
- nsites = number of times the motif is found in input sequences
- positive_hits = number of sequences in input containing at least 1 of the motif
- negative_hits = number of sequences in control containing at least 1 of the motif
- pval = p-value
- eval = E-value
- unerased_eval = Unerased E-Value
- positive_total = number of sequences in input
- negative_total = number of sequences in control
- pos_frac = fraction of positive sequences with a hit
- neg_frac = fraction of negative sequences with a hit
- motif = a universalmotif object of the discovered motif

## Citation

If you use `runDreme()` in your analysis, please cite:

Timothy L. Bailey, "DREME: Motif discovery in transcription factor ChIP-seq data", Bioinformatics, 27(12):1653-1659, 2011. full text

**Licensing:**

The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the MEME Suite Copyright Page for details.

## Examples

```
if (meme_is_installed()) {
# Create random named sequences as input for example
seqs <- universalmotif::create_sequences(rng.seed = 123)
names(seqs) <- seq_along(seqs)

# Runs dreme with default settings, shuffles input as background
runDreme(seqs, "shuffle")

# Runs searching for max 2 motifs, e-value cutoff = 0.1, explicitly using the DNA alphabet
runDreme(seqs, "shuffle", nmotifs = 2, e = 0.1, dna = TRUE)
}
```

---

# runFimo *Find instances of motifs using FIMO*

---

## Description

FIMO scans input sequences to identify the positions of matches to each input motif. FIMO has no sequence length or motif number restrictions.

## Usage

```
runFimo(
  sequences,
  motifs,
  bfile = "motif",
  outdir = "auto",
  parse_genomic_coord = TRUE,
  skip_matched_sequence = FALSE,
  max_strand = TRUE,
  text = TRUE,
  meme_path = NULL,
  silent = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| sequences | path to fasta file, or stringset input. |
| motifs | path to .meme format file, or universalmotif/universalmotif list input. |
| bfile | path to background file, or special values: "motif" to use 0-order frequencies contained in the motif, or "uniform" to use a uniform letter distribution. (default: "motif") |
| outdir | output directory location. Only used if text = FALSE. Default: "auto" to auto-generate directory name. Note: if not using a fasta path as input, this will be a temporary location unless explicity set. |
| parse_genomic_coord | logical(1) whether to parse genomic position from fasta headers. Fasta headers must be UCSC format positions (ie "chr:start-end"), but base 1 indexed (GRanges format). If names of fasta entries are genomic coordinates and parse_genomic_coord == TRUE, results will contain genomic coordinates of motif matches, otherwise FIMO will return relative coordinates (i.e. positions from 1 to length of the fasta entry). |
| skip_matched_sequence | logical(1) whether or not to include the DNA sequence of the match. Default: FALSE. Note: jobs will complete faster if set to TRUE. add_sequence() can be used to lookup the sequence after data import if parse_genomic_coord is TRUE, so setting this flag is not strictly needed. |
| max_strand | if match is found on both strands, only report strand with best match (default: TRUE). |
| text | logical(1) (default: TRUE). No output files will be created on the filesystem. The results are unsorted and no q-values are computed. This setting allows fast searches on very large inputs. When set to FALSE FIMO will discard 50% of the lower significance matches if >100,000 matches are detected. text = FALSE will also incur a performance penalty because it must first read a file to disk, then read it into memory. For these reasons, I suggest keeping text = TRUE. |
| meme_path | path to meme/bin/ (optional). Defaut: NULL, searches "MEME_PATH" environment variable or "meme_path" option for path to "meme/bin/". |
| silent | logical(1) whether to suppress stdout/stderr printing to console (default: TRUE). If the command is failing or giving unexpected output, setting silent = FALSE can aid troubleshooting. |
| ... | additional commandline arguments to fimo. See the FIMO Flag table below. |

## Details

Additional arguments passed to .... See: Fimo web manual for a complete description of FIMO flags.

| FIMO Flag | allowed values | default | description |
|---|---|---|---|
| alpha | numeric(1) | 1 | alpha for calculating position-specific priors. Rep |
| bfile | "motif", "motif-file", "uniform", file path, | "motif" | If "motif" or "motif-file", use 0-order letter freque |
| max_stored_scores | integer(1) | NULL | maximum number of scores to be stored for comp |
| motif_pseudo | numeric(1) | 0.1 | pseudocount added to motif matrix |

| | | | |
|---|---|---|---|
| no_qvalue | logical(1) | FALSE | only needed when text = FALSE, do not compute |
| norc | logical(1) | FALSE | Do not score reverse complement strand |
| prior_dist | file path | NULL | file containing binned distribution of priors |
| psp | file path | NULL | file containing position specific priors. Requires p |
| qv_thresh | logical(1) | FALSE | use q-values for the output threshold |
| thresh | numeric(1) | 1e-4 | output threshold for returning a match, only matc |

**Licensing:**
The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the MEME Suite Copyright Page for details.

## Value

GRanges object containing positions of each match. Note: if parse_genomic_coords = FALSE, each seqnames entry will be the full fasta header, and start/end will be the relative position within that sequence of the match. The GRanges object has the following additional mcols: * motif_id = primary name of matched motif * motif_alt_id = alternate name of matched motif * score = score of match (higher score is a better match) * pvalue = pvalue of the match * qvalue = qvalue of the match * matched_sequence = sequence that matches the motif

## Citation

If you use runFimo() in your analysis, please cite:

Charles E. Grant, Timothy L. Bailey, and William Stafford Noble, "FIMO: Scanning for occurrences of a given motif", Bioinformatics, 27(7):1017-1018, 2011. full text

## Examples

```
if (meme_is_installed()){
# Generate some example input sequences
seq <- universalmotif::create_sequences()
# sequences must have values in their fasta headers
names(seq) <- seq_along(seq)
# Create random example motif to search for
motif <- universalmotif::create_motif()

# Search for motif in sequences
# parse_genomic_coord set to FALSE since fasta headers aren't in "chr:start-end" format.
runFimo(seq, motif, parse_genomic_coord = FALSE)
}
```

---

# runMeme *Identify motifs with MEME*

---

## Description

MEME performs *de-novo* discovery of ungapped motifs present in the input sequences. It can be used in both discriminative and non-discriminative modes.

## Usage

```
runMeme(
  input,
  control = NA,
  outdir = "auto",
  alph = "dna",
  parse_genomic_coord = TRUE,
  combined_sites = FALSE,
  silent = TRUE,
  meme_path = NULL,
  ...
)

## S3 method for class 'list'
runMeme(
  input,
  control = NA,
  outdir = "auto",
  alph = "dna",
  parse_genomic_coord = TRUE,
  combined_sites = FALSE,
  silent = TRUE,
  meme_path = NULL,
  ...
)

## S3 method for class 'BStringSetList'
runMeme(
  input,
  control = NA,
  outdir = "auto",
  alph = "dna",
  parse_genomic_coord = TRUE,
  combined_sites = FALSE,
  silent = TRUE,
  meme_path = NULL,
  ...
)

## Default S3 method:
runMeme(
  input,
  control = NA,
  outdir = "auto",
  alph = "dna",
  parse_genomic_coord = TRUE,
  combined_sites = FALSE,
  silent = TRUE,
```

```
  meme_path = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| input | path to fasta, Biostrings::BStringSet list, or list of Biostrings::BStringSet (can generate using get_sequence()) |
| control | any data type as in input, or a character vector of names(input) to use those regions as control sequences. Using sequences as background requires an alternative objective function. Users must pass a non-default value of objfun to ... if using a non-NA control set (default: NA) |
| outdir | (default: "auto") Directory where output data will be stored. |
| alph | one of c("dna", "rna", "protein") or path to alphabet file (default: "dna"). |
| parse_genomic_coord | logical(1) whether to parse genomic coordinates from fasta headers. Requires headers are in the form: "chr:start-end", or will result in an error. Automatically set to FALSE if alph = "protein". This setting only needs to be changed if using a custom-built fasta file without genomic coordinates in the header. |
| combined_sites | logical(1) whether to return combined sites information (coerces output to list) (default: FALSE) |
| silent | Whether to suppress printing stdout to terminal (default: TRUE) |
| meme_path | path to "meme/bin/". If unset, will use default search behavior:<br>1. meme_path setting in options()<br>2. MEME_PATH setting in .Renviron or .bashrc |
| ... | additional arguments passed to MEME (see below) |

## Details

Note that MEME can take a long time to run. The more input sequences used, the wider the motifs searched for, and the more motifs MEME is asked to discover will drastically affect runtime. For this reason, MEME usually performs best on a few (<50) short (100-200 bp) sequences, although this is not a requirement. Additional details on how data size affects runtime can be found here.

MEME works best when specifically tuned to the analysis question. The default settings are unlikely to be ideal. It has several complex arguments documented here, which runMeme() accepts as R function arguments (see details below).

If discovering motifs within ChIP-seq, ATAC-seq, or similar peaks, MEME may perform best if using sequences flaking the summit (the site of maximum signal) of each peak rather than the center. ChIP-seq or similar data can also benefit from setting revcomp = TRUE, minw = 5, maxw = 20. For more tips on using MEME to analyze ChIP-seq data, see the following tips page.

### Additional arguments:

runMeme() accepts all valid arguments to meme as arguments passed to .... For flags without values, pass them as flag = TRUE. The dna, rna, and protein flags should instead be passed to the alph argument of runMeme(). The arguments passed to MEME often have many interactions with each other, for a detailed description of each argument see MEME Commandline Documentation.

## Value

MEME results in universalmotif_df format (see: `universalmotif::to_df()`). `sites_hits` is a nested data.frame column containing the position within each input sequence of matches to the identified motif.

## Citation

If you use `runMeme()` in your analysis, please cite:

Timothy L. Bailey and Charles Elkan, "Fitting a mixture model by expectation maximization to discover motifs in biopolymers", Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, pp. 28-36, AAAI Press, Menlo Park, California, 1994. pdf

## Licensing

The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the MEME Suite Copyright Page for details.

## Examples

```
if (meme_is_installed()) {
seqs <- universalmotif::create_sequences("CCRAAAW", seqnum = 4)
names(seqs) <- 1:length(seqs)
runMeme(seqs, parse_genomic_coord = FALSE)

}
```

---

# runStreme *Denovo motif discovery of target regions using STREME*

---

## Description

STREME discovers short, ungapped, *de-novo* motifs that are enriched or relatively enriched relative to a control set of sequences. STREME can be run to discover motifs relative to a shuffled set of input sequences, against a separately provided set of "control" sequences, or to determine whether motifs are centrally enriched within input sequences.

## Usage

```
runStreme(
  input,
  control,
  outdir = "auto",
  objfun = "de",
  alph = "dna",
  meme_path = NULL,
  silent = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| input | regions to scan for motifs. If using ‘objfun = "cd"‘ to test for centrally enriched motifs, be sure to include sufficient flanking sequence (e.g. +/- 500bp) for an accurate estimate. Can be any of: - path to fasta file - DNAStringSet object (can be generated from GRanges using ‘get_sequence()‘) - List of DNAStringSet objects (generated from ‘get_sequence()‘) - *NOTE:* if using StringSet inputs, each entry must be named (set with ‘names()‘). - *NOTE:* If you want to retain the raw streme output files, you must use a path to fasta file as input, or specify an "outdir" |
| control | regions to use as background for motif search. These should have a similar length distribution as the input sequences. Can be any of: - path to fasta file - DNAStringSet object (can be generated from GRanges using get_sequence) - A Biostrings::BStringSetList (generated using ‘get_sequence‘), in which case all sequences in the list will be combined as the control set. - if ‘input‘ is a list of DNAStringSet objects, a character vector of names in ‘input‘ will use those sequences as background. runstreme will not scan those regions as input. - "shuffle" to use streme’s built-in dinucleotide shuffle feature (NOTE: if ‘input‘ is a list object with an entry named "shuffle", the list entry will be used instead). Optionally can also pass ‘seed = <any number>‘ to ‘...‘ to use as the random seed during shuffling. If no seed is passed, streme will use 0 as the random seed, so results will be reproducible if rerunning. |
| outdir | path to output directory of streme files, or "auto" to autogenerate path. Default: location of input fasta in dir named "\<input\>_vs_\<control\>". If input is DNAstringset, will be temporary path. This means that if you want to save the raw output files, you must use fasta files as input or use an informative (and unique) outdir name. memes will **not check** if it overwrites files in a directory. Directories will be recursively created if needed. (default: "auto") |
| objfun | one of c("de", "cd"). Default: "de" for differential enrichment. "cd" for central distance (control must be set to NA for "cd"). |
| alph | one of c("dna", "rna", "protein") or a path to a MEME format alph file. (default: "dna") |
| meme_path | path to "meme/bin" |
| silent | Whether to suppress printing stdout & stderr to console (default: TRUE). Warnings are always printed regardless of this setting. |
| ... | pass any commandline options as R function arguments. For a complete list of STREME options, see [the STREME manual](https://meme-suite.org/meme/doc/streme.html). |

## Details

Properly setting the ‘control‘ parameter is key to discovering biologically relevant motifs. Often, using ‘control = "shuffle"‘ will produce a suboptimal set of motifs; however, some discriminative analysis designs don’t have proper "control" regions other than to shuffle.

If you have fewer than 50 sequences, consider using [runMeme()] instead.

# Citation

If you use ‘runStreme()‘ in your analysis, please cite:

Timothy L. Bailey, "STREME: Accurate and versatile sequence motif discovery", Bioinformatics, 2021. https://doi.org/10.1093/bioinformatics/btab203

# Licensing The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the [MEME Suite Copyright Page](http://meme-suite.org/doc/copyright.html) for details.

## Value

a ‘universalmotif_df‘ of STREME Motifs

## See Also

‘?universalmotif::tidy-motifs‘

---

`runTomTom` *Run TomTom on target motifs*

---

## Description

TomTom compares input motifs to a database of known, user-provided motifs to identify matches.

## Usage

```
runTomTom(
  input,
  database = NULL,
  outdir = "auto",
  thresh = 10,
  min_overlap = 5,
  dist = "ed",
  evalue = TRUE,
  silent = TRUE,
  meme_path = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `input` | path to .meme format file of motifs, a list of universalmotifs, or a universalmotif data.frame object (such as the output of `runDreme()`) |
| `database` | path to .meme format file to use as reference database (or list of universalmotifs). **NOTE:** p-value estimates are inaccurate when the database has fewer than 50 entries. |
| `outdir` | directory to store tomtom results (will be overwritten if exists). Default: location of input fasta file, or temporary location if using universalmotif input. |

| | |
|---|---|
| thresh | report matches less than or equal to this value. If evalue = TRUE (default), set an e-value threshold (default = 10). If evalue = FALSE, set a value between 0-1 (default = 0.5). |
| min_overlap | only report matches that overlap by this value or more, unless input motif is shorter, in which case the shorter length is used as the minimum value |
| dist | distance metric. Valid arguments: allr \| ed \| kullback \| pearson \| sandelin \| blic1 \| blic5 \| llr1 \| llr5. Default: ed (euclidean distance). |
| evalue | whether to use E-value as significance threshold (default: TRUE). If evalue = FALSE, uses *q-value* instead. |
| silent | suppress printing stderr to console (default: TRUE). |
| meme_path | path to "meme/bin/" (optional). If unset, will check R environment variable "MEME_DB (set in .Renviron), or option "meme_db" (set with option(meme_db = "path/to/meme/bin")) |
| ... | additional flags passed to tomtom using cmdfun formating (see table below for details) |

## Details

runTomTom will rank matches by significance and return a best match motif for each input (whose properties are stored in the best_match_* columns) as well as a ranked list of all possible matches stored in the tomtom list column.

Additional arguments

runTomTom() can accept all valid tomtom arguments passed to ... as described in the tomtom commandline reference. For convenience, below is a table of valid arguments, their default values, and their description.

| TomTom Flag | allowed values | default | description |
|---|---|---|---|
| bfile | file path | NULL | path to background model for converting frequency matrix to log-odds score |
| motif_pseudo | numeric | 0.1 | pseudocount to add to motifs |
| xalph | logical | FALSE | convert alphabet of target database to alphabet of query database |
| norc | logical | FALSE | Do not score reverse complements of motifs |
| incomplete_scores | logical | FALSE | Compute scores using only aligned columns |
| thresh | numeric | 0.5 | only report matches with significance values <= this value. Unless evalue = |
| internal | logical | FALSE | forces the shorter motif to be completely contained in the longer motif |
| min_overlap | integer | 1 | only report matches that overlap by this number of positions or more. If quer |
| time | integer | NULL | Maximum runtime in CPU seconds (default: no limit) |

## Value

data.frame of match results. Contains best_match_motif column of universalmotif objects with the matched PWM from the database, a series of best_match_* columns describing the TomTom results of the match, and a tomtom list column storing the ranked list of possible matches to each motif. If a universalmotif data.frame is used as input, these columns are appended to the data.frame. If no matches are returned, tomtom and best_match_motif columns will be set to NA and a message indicating this will print.

## Citation

If you use `runTomTom()` in your analysis, please cite:

Shobhit Gupta, JA Stamatoyannopolous, Timothy Bailey and William Stafford Noble, "Quantifying similarity between motifs", Genome Biology, 8(2):R24, 2007. full text

**Licensing:**

The MEME Suite is free for non-profit use, but for-profit users should purchase a license. See the MEME Suite Copyright Page for details.

## Examples

```
if (meme_is_installed()) {
motif <- universalmotif::create_motif("CCRAAAW")
database <- system.file("extdata", "flyFactorSurvey_cleaned.meme", package = "memes")

runTomTom(motif, database)
}
```

---

`update_best_match` *Update best match info by ranking of tomtom data*

---

## Description

This function updates the best_match columns based on the rankings on the tomtom list data. By re-ordering the entries of a ‘tomtom‘ object, the best_match columns can be updated to reflect the new rankings using [update_best_match()], where the first row of the ‘tomtom‘ data.frame is selected as the best match.

## Usage

```
update_best_match(res)
```

## Arguments

| | |
|---|---|
| `res` | results from runTomTom |

## Value

‘res‘ with updated best_* columns

## See Also

[force_best_match()]

## Examples

```
data("example_dreme_tomtom")
# best match is "CG2052_SANGER_2.5"
example_dreme_tomtom$best_match_name[1]
# reorder the `tomtom` data.frame
example_dreme_tomtom$tomtom[[1]] <- dplyr::arrange(example_dreme_tomtom$tomtom[[1]],
                                                   dplyr::desc(match_eval))
# update_best_match will use the new order of rows, taking the top row as the new best match
new_res <- update_best_match(example_dreme_tomtom)
# best match is now altered:
new_res$best_match_name[1]
```

---

# view_tomtom_hits *Compare top tomtom hits to original motif*

---

## Description

Although TomTom does a good job of matching unknown motifs to known motifs, sometimes the top hit is not the correct assignment. It can be useful to manually inspect the hits. This function provides a quick utility to compare matches.

## Usage

```
view_tomtom_hits(results, top_n = "all")
```

## Arguments

| | |
|---|---|
| results | results data.frame from [runTomTom()] |
| top_n | number of matched motifs to return in plot (default: "all") |

## Details

This is intended to be a function used interactively and may not always be the best tool for creating publication-quality figures. Results with matches return ggseqlogo outputs which can be further manipulated using [ggplot2::theme()] calls, but results containing no matches are static plots.

## Value

plot of input motif vs the top n number of tomtom matched motifs. If no match found, will plot "No Match". Note: the "No Match" plots are not amenable to ggplot theme() manipulations, while all others are.

## Examples

```
results <- importTomTomXML(system.file("extdata", "tomtom.xml", package = "memes"))
# show top 3 hits
view_tomtom_hits(results, top_n = 3)
```

`write_fasta` *Write fasta file from stringset*

## Description

Write fasta file from stringset

## Usage

```
write_fasta(seq, path = tempfile(fileext = ".fa"))
```

## Arguments

| | |
|---|---|
| `seq` | a 'Biostrings::XStringSet' |
| `path` | path of fasta file to write (default: temporary file) |

## Value

path to created fasta file

## Examples

```
seq <- universalmotif::create_sequences()

write_fasta(seq)
```

# Index