Package: tailor (via r-universe)

June 9, 2026

Type Package

Title Tool for Adapter-domain Identification and Linking Of RBPs

Version 0.1.1

Description Bacteriophages (phages) are viruses that infect bacteria. Many phages encode receptor binding proteins (RBPs) that mediate attachment to their bacterial hosts. Most RBPs contain a conserved N-terminal domain (we refer to this as an ``adapter'') which anchors the protein to the phage particle. This package provides functions to indentify these adapters and group them based on sequence similarity.

Depends R (>= 4.1.0)

Imports breakpoint, doParallel, dplyr, foreach, ggplot2, patchwork, pwalign, qcc, rlang, reshape2, tibble, tidyr

Suggests covr, testthat, vdiffr

Remotes stitam/breakpoint

License GPL (>= 3)

Encoding UTF-8

LazyData true

Config/testthat/edition 3

Config/roxygen2/version 8.0.0.9000

Config/pak/sysreqs libicu-dev zlib1g-dev

Repository https://stitam.r-universe.dev

Date/Publication 2026-06-09 08:37:49 UTC

RemoteUrl https://github.com/sbthandras/tailor

RemoteRef HEAD

RemoteSha 3929b27e63c6e52248d5d610b9937290d7da7596

Contents

`adapter_matrix` *Create adapter matrix*

Description

Convert identified adapters into matrix form.

Usage

```
adapter_matrix(
  adapters,
  ids = NULL,
  value = "pident",
  verbose = getOption("verbose")
)
```

Arguments

`adapters`	adapter; a data frame of class "adapter" containing identified adapter sequences.
`ids`	character; ids within the 'adapters' data frame, in the same order as they should appear in the matrix. If NULL (default), order will be determined by the order of unique ids in the 'adapters' data frame. If specified, all ids must be present in the'adapters' data frame and all ids in the 'adapters' data frame must be present in the 'ids' vector.
`value`	character; the variable within the adapter data frame to use as values in the matrix, "pident", "mean_score", or "end".
`verbose`	logical; should verbose messages be printed to the console? If TRUE, a progress bar is also displayed.

Note

The function handles incomplete adapter data frames where some sequence pairs are missing. Missing pairs are assumed to lack shared adapters and are filled with 0 values in the matrix. However, sequences with no shared adapters across any pair are not recoverable if they were dropped from the data frame.

Examples

```
# import example data
data(rbps)
# import adapters identified from example data
data(adapters)
# convert to matrix and plot
adapters |> adapter_matrix() |> plot()
# use same order as in the original data frame
adapters |> adapter_matrix(ids = rbps$Core_ORF) |> plot()
```

`adapters` *Adapters*

Description

This data set contains identified adapters from all pairwise comparisons of the RBPs in the ‘rbps‘ data set. The approach to calculate these adapters was the following: first, check for adapters with the "xbar.one" method, because it seems to be highly specific (i.e. it rarely identifies adapters when there isn’t one). If an adapter was found, then the "cemean" method was used to get a more accurate estimate of the adapter length.

Usage

```
adapters
```

Format

A data frame with 90 rows and 6 variables:

pattern_id pattern_id

subject_id subject_id

start location of the first amino acid within the adapter sequence

end location of the last amino acid within the adapter sequence

mean_score mean amino acid position score between ’start’ and ’end’

pident ratio of identical amino acids between ’start’ and ’end’

`cluster_adapters` *Group RBPs into adapter clusters*

Description

Group RBPs into adapter clusters

Usage

```
cluster_adapters(
  mat,
  k_min = 1,
  k_max = NULL,
  h = NULL,
  homogeneity_thr = 0.75,
  completeness_thr = 0.75,
  cores = 1,
  verbose = getOption("verbose")
)
```

Arguments

`mat`	adapter_matrix; a matrix of class "adapter_matrix". Use adapter_matrix() to create a compatible matrix.
`k_min`	integer; minimum number of clusters to split the RBPs into.
`k_max`	integer; maximum number of clusters to split the RBPs into.
`h`	hclust object; a hierarchical clustering object created from the distance matrix. If NULL, the function will create a hierarchical clustering object from the input matrix using the average method.
`homogeneity_thr`	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter for homogeneity.
`completeness_thr`	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter for completeness.
`cores`	integer; number of CPU cores to use.
`verbose`	logical; should verbose messages be printed to the console? If TRUE, a progress bar is also displayed.

Value

a data frame with two columns, id and cluster, containing cluster assignments.

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# cluster RBPs into 2 to 5 clusters
cluster_adapters(amat, k_min = 2, k_max = 5)
```

`completeness` *Completeness*

Description

Completeness measures measures how much similar samples are put together by the clustering algorithm. In the context of adapter sequences of RBPs, completeness measures whether RBPs that share adapters with members of a cluster are also included within that cluster. High completeness score for a cluster indicates that when cluster members share adapters with other RBPs, those other RBPs tend to be in the same cluster as well. On the other hand, low scores suggests that cluster members often share adapters with RBPs that do not belong to the cluster.

Usage

```
completeness(mat, index, threshold = 0.75)
```

Arguments

`mat`	adapter_matrix; a matrix of class "adapter_matrix". Use adapter_matrix() to create a compatible matrix.
`index`	numeric; indices of proteins that are included within a cluster.
`threshold`	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter.

Value

A numeric value between 0 and 1. A value of 1 indicates that all proteins sharing features with cluster members are contained within the cluster, while a value of 0 indicates that shared features are found equally inside and outside the cluster.

Note

Note that `mat` is different for homogeneity and completeness. When calculating homogeneity, we are only interested in comparisons within a clusters and so `mat` represents a cluster. However, when calculating completeness, we are also interested in comparisons outside a cluster, and so `mat` must contain all comparisons, not just the cluster in question.

References

https://medium.com/data-science/v-measure-an-homogeneous-and-complete-clustering-ab5b1823d0ad

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# split to two clusters
clusters <- cluster_adapters(amat, k_min = 2, k_max = 2)
# calculate completeness for ACL 1
ids <- clusters |> dplyr::filter(cluster == "ACL 1") |> dplyr::pull(id)
index <- which(rownames(amat) %in% ids)
completeness(mat = amat, index = index)
```

---

filter_comparisons *Filter comparisons*

Description

Filter comparisons (breakpoints or adapters) based on ids or properties from the supporting data set.

Usage

```
filter_comparisons(comps, filter_value, filter_by = "Core_ORF", data = NULL)
```

Arguments

comps	data.frame; the comparisons data set to be filtered. Must be of class "breakpoints" or "adapter". See find_breakpoints, find_adapter or find_all_adapters for compatible data frames.
filter_value	character; the value to filter by. By default, this is a pattern_id or subject id from the comparisons data frame, but can be any value in the supporting data set if filter_by and data are specified.
filter_by	character; name of the variable in the supporting data set to filter by. By default, this is "Core_ORF", but can be any column in the supporting data set.
data	data.frame; the supporting data set that links ids with properties. This is required if filter_by is not "Core_ORF". Must contain a column with the same ids as the comparisons data frame (e.g. "Core_ORF") and a column with the values to filter by (e.g. "ACL 2").

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# cluster RBPs into 2 to 5 clusters
clusters <- cluster_adapters(amat, k_min = 2, k_max = 5)
rbps <- dplyr::left_join(rbps, clusters, by = c("Core_ORF" = "id"))
```

```
# adapters with "ON513429-1"
adapters |> filter_comparisons("ON513429-1")
# adapter between "MN395291-1" and "ON513429-1"
adapters |> filter_comparisons("MN395291-1") |> filter_comparisons("ON513429-1")
# adapters in ACL 2
adapters |> filter_comparisons("ACL 2", filter_by = "cluster", data = rbps)
```

`find_adapter` *Find adapter region*

Description

This functions takes a data frame of breakpoints from a pairwise global alignment and detects conserved N-terminal regions, optinally merging neighboring conserved regions.

Usage

```
find_adapter(
  bps,
  min_pident = 0.4,
  max_start = 10,
  merge_beginning = TRUE,
  merge_conserved = TRUE
)
```

Arguments

`bps`	data.frame; a data frame of breakpoints
`min_pident`	numeric; the lowest accepted percentage identity for a region to be considered a "conserved" region.
`max_start`	integer; the maximum accepted starting position for the first conserved region.
`merge_beginning`	logical; whether to merge the positions before the first conserved region. See Details for more information.
`merge_conserved`	logical; whether neighboring conserved regions should be merged. See Details for more information.

Details

The function returns a single row for each pairwise comparison. If no conserved N-terminal regions are identified, the function returns a row with NA values for the start, end, mean_score, and pident columns. If a single conserved N_terminal region is identified, the function returns the start and end positions, mean score, and percentage identity of this region. If multiple conserved N-terminal regions are identified, behaviour depends on the values of ‘merge_conserved‘ and ‘merge_beginning‘.

If ‘merge_conserved‘ is TRUE, neighboring conserved regions will be merged and the mean score and percentage identity of the merged region will be recalculated. If ‘merge_conserved‘ is FALSE, the function will only keep the first conserved region.

If ‘merge_beginning‘ is TRUE, and there is a conserved region (merged or unmerged) which starts within ‘max_start‘, but not at position 1, then the function will also merge the positions before this conserved region and recalculate the mean score and percentage identity of the merged region. If the percentage identity of the merged region is below ‘min_pident‘, the function will drop the merged region and return NA values for the start, end, mean_score, and pident columns.

Value

A data frame of class "adapter".

Examples

```
data(rbps)
ps <- position_scores("MN395291-1", "ON513429-1", data = rbps)
bps <- find_breakpoints(ps, Nmax = 5)
find_adapter(bps)
```

---

find_all_adapters *Find all adapters for a set of RBPs*

Description

Look at all pairs between a set of RBPs, align, detect breakpoints, find adapters and combine them into a single data frame.

Usage

```
find_all_adapters(
  ids,
  data,
  id_var = "Core_ORF",
  seq_var = "translation",
  submat = "BLOSUM80",
  max_end_vars = NULL,
  max_end = NULL,
  method = "cemean",
  min_pident = 0.4,
  max_start = 10,
  merge_beginning = TRUE,
  merge_conserved = TRUE,
  cores = 1,
  verbose = getOption("verbose"),
  ...
)
```

Arguments

ids	character; the names of RBPs to compare.
data	data.frame; a data frame that contains IDs and sequences
id_var	character; variable within data that stores IDs.
seq_var	character; variable within data that stores sequences.
submat	character; the substitution matrix. See position_scores() for more information.
max_end_vars	character; optional vector of variable names in data that contain numeric values representing positions which the adapters should not overlap with and terminate earlier (e.g., domain start positions). See Details for more information.
max_end	integer; the maximum position to look for breakpoints. If 'NULL', search the full alignment, if an integer, limit the search to that position. If both this argument and 'position_scores$max_end' are set, the smaller integer is used.
method	character; the method to use for finding breakpoints. Either "cemean", "plateau", or "window". See find_breakpoints() for more information
min_pident	numeric; the lowest accepted percentage identity for a region to be considered a "conserved" region.
max_start	integer; the maximum accepted starting position for the first conserved region.
merge_beginning	logical; whether to merge the positions before the first conserved region. See Details for more information.
merge_conserved	logical; whether neighboring conserved regions should be merged. See Details for more information.
cores	integer; number of CPU cores to use.
verbose	logical; should verbose messages be printed to the console?
...	additional arguments to pass to breakpoint detection. See find_breakpoints() for more information.

Details

If you provide one or more 'max_end_var' names, the function will look for their values in both the pattern and subject rows, calculate their minimum and return it in the output object. The value will then be respected by 'find_breakpoints()' and the function will not look at positions beyond this value.

Value

A data.frame of class "adapter".

Note

When comparing all pairs of sequences, most pairs will not share an adapter, resulting in a sparse data frame. It makes sense to remove NA rows and export only those with shared adapters. The 'adapter_matrix()' function assumes missing pairs did not have a shared adaptor and restores these pairs with 0 values in the matrix. However, sequences with no shared adapters to any other sequence are dropped from the data frame and cannot be recovered in the matrix.

Examples

```
data(rbps)
find_all_adapters(rbps$Core_ORF[1:3], data = rbps)
```

`find_breakpoints` *Find breakpoints in global alignment scores*

Description

This function looks at global alignment scores by position and attempts to find locations along the length of the aligned sequences where these scores tend to change substantially.

Usage

```
find_breakpoints(position_scores, max_end = NULL, method = "cemean", ...)

find_breakpoints_cemean(position_scores, Nmax = 5, seed = 0)

find_breakpoints_plateau(position_scores, pinit = 0.05, type = "ewma", ...)

find_breakpoints_window(
  position_scores,
  window = 5,
  score_threshold = 3,
  pident_threshold = 0.4,
  p_threshold = 0.05
)
```

Arguments

`position_scores`	an object of class ‘ps‘, returned by ‘position_scores()‘.
`max_end`	integer; the maximum position to look for breakpoints. If ‘NULL‘, search the full alignment, if an integer, limit the search to that position. If both this argument and ‘position_scores$max_end‘ are set, the smaller integer is used.
`method`	character; the method to use for finding breakpoints. Either "cemean", "plateau", or "window".
`...`	additional arguments depending on the selected method. When ‘method = "plateau"‘, additional arguments passed on to the selected control chart. See ‘?qcc::cusum()‘, ‘?qcc::ewma()‘, or ‘?qcc::qcc()‘ for a full list of control chart parameters.
`Nmax`	integer; the maximum number of breakpoints to look for.
`seed`	integer; random seed.
`pinit`	numeric; ratio of sequence length used for setting the center. See Details for more information.

`type`	character; the type of statistical process control chart to use. Currently supported cards are ‘cusum‘, ‘ewma‘ and ‘xbar.one‘.
`window`	integer; the number of successive amino acids to look at
`score_threshold`	numeric; mu threshold for alignment scores
`pident_threshold`	numeric; mu threshold for amino acid identities
`p_threshold`	numeric; p value threshold of significance

Details

When using ‘method = "cemean"‘ the function will estimate the number of breakpoints and their locations using the Cross-Entropy Method. This method will use the ‘CE.Normal.Mean()‘ function from the ‘breakpoint‘ package to find the breakpoints.

When using ‘method = "plateau"‘ the function will look at the first few positions of the alignment (governed by ‘pinit‘) and calculate a mean score. This mean score will be used as the center: 1. it will be used as the center point for a number of control charts; 2. any scores above the center will be reduced to be identical with the center. Then the function will use the control chart selected by ‘type‘, and extract sections using the violations.

When using ‘method == "window"‘ the function will form disjunct windows along the length of the alignment and look at the positions within each window. For each window, it will perform one sided wilcoxon tests against the ‘score_threshold‘ and the ‘pident_threshold‘ and consider the window conserved if at least one of the tests passes. Finally it will collapse neighboring windows that are all considered conserved or non-conserved and return a data frame of sections in which conservation alternates between the lines.

Value

A data frame with 6 columns:

- ‘pattern_id‘: pattern ID
- ‘subject_id‘: subject ID
- ‘start‘: first position for a section of the alignment
- ‘end‘: last position for a section of the alignment
- ‘mean_score‘: mean score for amino acid pairs within the section
- ‘pident‘: ratio of matching amino acid pairs within the section

Note

When using ‘method == "cemean"‘ the speed of the function will depend on the number of breakpoints to look for.

See Also

[position_scores()]

Examples

```
data(rbps)
ps <- position_scores("MN395291-1", "ON513429-1", data = rbps)

# Find breakpoints using the Cross-Entropy Method
find_breakpoints(ps, method = "cemean", Nmax = 5)
# Find breakpoints using plateau on scores and EWMA chart
find_breakpoints(ps, method = "plateau", type = "ewma", lambda = 0.2)
# Find breakpoints using plateau on scores and CUSUM chart
find_breakpoints(ps, method = "plateau", type = "cusum")
# Find breakpoints using disjunct windows
find_breakpoints(ps, method = "window", window = 5)
```

`homogeneity` *Homogeneity*

Description

Homogeneity measures how much the samples in a group are similar. The function calculates the percentage of pairs that are similar within the group, relative to the total number of pairs in the group. In the context of adapter sequences of RBPs, homogeneity measures the similarity of adapter sequences within a cluster of RBPs. A high homogeneity score for the cluster indicates that the RBPs in the cluster tend to have similar adapter sequences, while a low score suggests that the adapter sequences are more diverse within the cluster.

Usage

```
homogeneity(mat, threshold = 0.75)
```

Arguments

`mat`	adapter_matrix; a matrix of class "adapter_matrix". Use adapter_matrix() to create a compatible matrix.
`threshold`	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter.

Value

A numeric value between 0 and 1. A value of 1 indicates that all samples in the group are similar, while a value of 0 indicates that no samples in the group are similar.

Note

Note that `mat` is different for homogeneity and completeness. When calculating homogeneity, we are only interested in comparisons within a clusters and so `mat` represents a cluster. However, when calculating completeness, we are also interested in comparisons outside a cluster, and so `mat` must contain all comparisons, not just the cluster in question.

References

https://medium.com/data-science/v-measure-an-homogeneous-and-complete-clustering-ab5b1823d0ad

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# split to two clusters
clusters <- cluster_adapters(amat, k_min = 2, k_max = 2)
# calculate homogeneity for ACL 1
ids <- clusters |> dplyr::filter(cluster == "ACL 1") |> dplyr::pull(id)
index <- which(rownames(amat) %in% ids)
homogeneity(mat = amat[index, index])
```

---

plot.adapter_matrix *Plot adapter matrix*

Description

Take an adapter matrix and optionally data frame of cluster designations and plot them on a simple heatmap.

Usage

```
## S3 method for class 'adapter_matrix'
plot(x, clusters = NULL, ...)
```

Arguments

x	adapter_matrix; a matrix of class "adapter_matrix". Use adapter_matrix() to create a compatible matrix.
clusters	data.frame; a table of RBP IDs and cluster designations.
...	additional arguments (not used).

Value

A heatmap. If clusters are provided, also show cluster assignments.

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# plot pident matrix
plot(amat)
```

```
# split RBPs into two clusters
clusters <- cluster_adapters(amat, k_min = 2, k_max = 2)
# plot pident matrix with cluster assignments
plot(amat, clusters = clusters)
```

---

plot.ps *Plot pairwise alignment scores with adapter regions*

Description

This is a convenience function for looking at pairwise alignments. The function will plot alignment scores along the length of the alignment. At the same time, the function will also attempt to find an adapter region using various approaches and add the end of the conserved region to the plot.

Usage

```
## S3 method for class 'ps'
plot(x, type = "ma", method = NULL, highlight = NULL, size = 10, ...)
```

Arguments

x	an object of class ‘ps‘, returned by ‘position_scores()‘.
type	character; strategy for plotting data points. Either "indiv", "ma", or "cusum". See Details for more information.
method	character; the method to use for finding the conserved N terminal region. Can be one or more of "cemean", "cusum", "ewma", "xbarone", and "window".
highlight	character; the method to use for highlighting the conserved N terminal region. Must be one of the methods used in ‘method‘.
size	integer; the number of successive data point to use for calculating moving averages. Only used if ‘type == "ma"‘.
...	additional arguments (not used).

Details

When ‘type = "indiv"‘ each alignment score will be plotted separately. When ‘type = "ma"‘ the plot will show moving averages. When ‘type = "cusum"‘ the plot will show the cumulative sum of scores by position.

Value

a plot.

Examples

```
data(rbps)
ps <- position_scores("MN395291-1", "ON513429-1", data = rbps)
plot(ps)
```

position_scores *Pairwise global alignments with position scores*

Description

This function takes a pair of IDs within a data set, performs a pairwise global alignment on their sequences, and returns an object which contains the pattern and subject IDs and a tibble of nucleotide or amino acid identities and alignment scores by position.

Usage

```
position_scores(
  pattern_id,
  subject_id,
  data,
  id_var = "Core_ORF",
  seq_var = "translation",
  submat = "BLOSUM80",
  max_end_vars = NULL,
  verbose = getOption("verbose")
)
```

Arguments

pattern_id	character; pattern ID
subject_id	character; subject ID
data	data.frame; a data frame that contains IDs and sequences
id_var	character; variable within data that stores IDs.
seq_var	character; variable within data that stores sequences.
submat	character; the substitution matrix. Options include "BLOSUM45", "BLOSUM50", "BLOSUM62", "BLOSUM80", "BLOSUM100", "PAM30", "PAM40", "PAM70", "PAM120", "PAM250".
max_end_vars	character; optional vector of variable names in data that contain numeric values representing positions which the adapters should not overlap with and terminate earlier (e.g., domain start positions). See Details for more information.
verbose	logical; should verbose messages be printed to the console?

Details

If you provide one or more ‘max_end_var‘ names, the function will look for their values in both the pattern and subject rows, calculate their minimum and return it in the output object. The value will then be respected by ‘find_breakpoints()‘ and the function will not look at positions beyond this value.

Value

an object of class ps which is a list with five elements: pattern_id, subject_id, position_scores and substitution_matrix, max_end. position scores is a tibble that contains nucleotide or amino acid identities and alignment scores by position.

See Also

find_breakpoints()

Examples

```
data(rbps)
position_scores("MN395291-1", "ON513429-1", data = rbps)
```

---

rbps *Receptor Binding Proteins*

Description

This data set contains example bacteriophage receptor binding proteins (RBPs) for demonstration purposes.

Usage

```
rbps
```

Format

A data frame with 20 rows and 2 variables:

Core_ORF Unique identifier for each receptor binding protein

translation Amino acid sequence of the protein

---

schoco *SCores of HOmogeneity and COmpleteness*

Description

Calculate homogeneity and completeness scores for clusters of RBPs based on the percentage identity of shared adapter sequences. The "schoco" score is a weighted sum of the homogeneity and completeness scores, where homogeneity is weighted by the size of the cluster and completeness is weighted by the log2 of the size of the cluster.

Usage

```
schoco(
  mat,
  k,
  h = NULL,
  homogeneity_thr = 0.75,
  completeness_thr = 0.75,
  verbose = getOption("verbose")
)
```

Arguments

mat	adapter_matrix; a matrix of class "adapter_matrix". Use adapter_matrix() to create a compatible matrix.
k	integer; the number of clusters to split the RBPs into.
h	hclust object; a hierarchical clustering object created from the distance matrix. If NULL, the function will create a hierarchical clustering object from the input matrix using the average method.
homogeneity_thr	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter for homogeneity.
completeness_thr	numeric; minimum percentage identity of the identified adapter sequence to be considered a shared adapter for completeness.
verbose	logical; should verbose messages be printed to the console?

Value

A data frame with 6 columns: nclust (number of clusters requested), cluster (cluser identifier), homogeneity (homogeneity score for the cluster), completeness (completeness score for the cluster), schoco (sum of weighted homogeneity and completeness scores, where homogeneity is multiplied by the size of the cluster and completeness is multiplied by log2 of the size of the cluster. The data frame contains one row for each cluster.

Examples

```
data(rbps)
data(adapters)
# convert to pident matrix and order by the original data frame
amat <- adapter_matrix(adapters, ids = rbps$Core_ORF)
# split matrix into two clusters and calculate schoco values
schoco(amat, k = 2)
```

validate_rbps *Validate RBP data set*

Description

This function can be used to check the RBP data set for any consistency issues that would produce errors in downstream analysis.

Usage

```
validate_rbps(
  rbps,
  id_var = "Core_ORF",
  seq_var = "translation",
  verbose = getOption("verbose")
)
```

Arguments

rbps	data.frame
id_var	name of the variable in rbps that contains unique identifiers for each RBP. Must be unique and of class "character".
seq_var	name of the variable in rbps that contains the amino acid sequences. Must be of class "character" and cannot contain empty strings or unknown characters.
verbose	should the function return verbose messages to console?

Details

The function checks whether id_var is present in the data set and is unique, and whether seq_var is present in the data set, is of class "character", and does not contain empty strings. The function also checks whether seq_var contains unknown characters, i.e. characters other than the 20 standard amino acids and the ambiguous characters B, Z, and X. If any of these checks fail, the function returns FALSE and, if verbose = TRUE, also returns a warning message to console.

Value

The function returns TRUE if the tails data set is consistent and returns FALSE if it is not.

Examples

```
data(rbps)
validate_rbps(rbps, verbose = TRUE)
```

Index