Package 'aphid'

January 23, 2026

Type Package

Title Analysis with Profile Hidden Markov Models

Version 1.3.6

Author Shaun Wilkinson [aut, cre]

Maintainer Shaun Wilkinson <shaunpwilkinson@gmail.com>

Description Designed for the development and application of
hidden Markov models and profile HMMs for biological sequence analysis.
Contains functions for multiple and pairwise sequence alignment,
model construction and parameter optimization, file import/export,
implementation of the forward, backward and Viterbi algorithms for
conditional sequence probabilities, tree-based sequence weighting,
and sequence simulation.
Features a wide variety of potential applications including
database searching, gene-finding and annotation, phylogenetic
analysis and sequence classification.
Based on the models and algorithms described in Durbin et
al (1998, ISBN: 9780521629713).

License GPL-3

URL https://github.com/shaunpwilkinson/aphid

BugReports https://github.com/shaunpwilkinson/aphid/issues

LazyData TRUE

Encoding UTF-8

SystemRequirements GNU make

Depends R(>= 3.0.0)

Imports graphics, openssl, kmer (>= 1.0.0), qpdf, Rcpp (>= 0.12.5),
stats

Suggests ape (>= 4.0), knitr, rmarkdown, testthat

LinkingTo Rcpp

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-01-23 06:41:06 UTC

Contents

`align` *Multiple sequence alignment in R.*

Description

`align` performs a multiple alignment on a list of sequences using profile hidden Markov models.

Usage

```
align(x, ...)

## S3 method for class 'DNAbin'
align(
  x,
  model = NULL,
  progressive = FALSE,
  seeds = NULL,
  seqweights = "Henikoff",
  refine = "Viterbi",
  k = 5,
  maxiter = 100,
  maxsize = NULL,
  inserts = "map",
  lambda = 0,
  threshold = 0.5,
  deltaLL = 1e-07,
  DI = FALSE,
  ID = FALSE,
  residues = NULL,
  gap = "-",
  pseudocounts = "background",
  qa = NULL,
  qe = NULL,
  cores = 1,
  quiet = FALSE,
  ...
)

## S3 method for class 'AAbin'
align(
  x,
  model = NULL,
  progressive = FALSE,
  seeds = NULL,
  seqweights = "Henikoff",
  refine = "Viterbi",
  k = 5,
  maxiter = 100,
  maxsize = NULL,
  inserts = "map",
  lambda = 0,
  threshold = 0.5,
  deltaLL = 1e-07,
  DI = FALSE,
  ID = FALSE,
  residues = NULL,
  gap = "-",
  pseudocounts = "background",
  qa = NULL,
  qe = NULL,
  cores = 1,
  quiet = FALSE,
  ...
```

```
)

## S3 method for class 'list'
align(
  x,
  model = NULL,
  progressive = FALSE,
  seeds = NULL,
  seqweights = "Henikoff",
  k = 5,
  refine = "Viterbi",
  maxiter = 100,
  maxsize = NULL,
  inserts = "map",
  lambda = 0,
  threshold = 0.5,
  deltaLL = 1e-07,
  DI = FALSE,
  ID = FALSE,
  residues = NULL,
  gap = "-",
  pseudocounts = "background",
  qa = NULL,
  qe = NULL,
  cores = 1,
  quiet = FALSE,
  ...
)

## Default S3 method:
align(
  x,
  model,
  pseudocounts = "background",
  residues = NULL,
  gap = "-",
  maxsize = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

x	a list of DNA, amino acid, or other character sequences consisting of symbols emitted from the chosen residue alphabet. The vectors can either be of mode "raw" (consistent with the "DNAbin" or "AAbin" coding scheme set out in the ape package), or "character", in which case the alphabet should be specified in the `residues` argument. This argument can alternatively be a vector representing a single sequence. In this case, and if the second argument is also a single

	sequence, a standard pairwise alignment is returned.
`...`	aditional arguments to be passed to `"Viterbi"` (if `refine = "Viterbi"`) or `"forward"` (if `refine = "BaumWelch"`).
`model`	an optional profile hidden Markov model (a `"PHMM"` object) to align the sequences to. If `NULL` a PHMM will be derived from the list of sequences, and each sequence will be aligned back to the model to produce the multiple sequence alignment.
`progressive`	logical indicating whether the alignment used to derive the initial model parameters should be built progressively (assuming input is a list of unaligned sequences, ignored otherwise). Defaults to FALSE, in which case the longest sequence or sequences are used (faster, but possibly less accurate).
`seeds`	optional integer vector indicating which sequences should be used as seeds for building the guide tree for the progressive alignment (assuming input is a list of unaligned sequences, and `progressive = TRUE`, ignored otherwise). Defaults to NULL, in which a set of log(n, 2)^2 non-identical sequences are chosen from the list of sequences by k-means clustering.
`seqweights`	either NULL (all sequences are given weights of 1), a numeric vector the same length as `x` representing the sequence weights used to derive the model, or a character string giving the method to derive the weights from the sequences (see `weight`).
`refine`	the method used to iteratively refine the model parameters following the initial progressive alignment and model derivation step. Current supported options are `"Viterbi"` (Viterbi training; the default option), `"BaumWelch"` (a modified version of the Expectation-Maximization algorithm), and "none" (skips the model refinement step).
`k`	integer representing the k-mer size to be used in tree-based sequence weighting (if applicable). Defaults to 5. Note that higher values of k may be slow to compute and use excessive memory due to the large numbers of calculations required.
`maxiter`	the maximum number of EM iterations or Viterbi training iterations to carry out before the cycling process is terminated and the partially trained model is returned. Defaults to 100.
`maxsize`	integer giving the upper bound on the number of modules in the PHMM. If NULL no maximum size is enforced.
`inserts`	character string giving the model construction method in which alignment columns are marked as either match or insert states. Accepted methods include `"threshold"` (only columns with fewer than a specified proportion of gaps form match states in the model), `"map"` (default; match and insert columns are found using the maximum *a posteriori* method outlined in Durbin et al (1998) chapter 5.7), `"inherited"` (match and insert columns are inherited from the input alignment), and `"none"` (all columns are assigned match states in the model). Alternatively, insert columns can be specified manually by providing a logical vector the same length as the number of columns in the alignment, with `TRUE` for insert columns and `FALSE` for match states.

`lambda`	penalty parameter used to favour models with fewer match states. Equivalent to the log of the prior probability of marking each column (Durbin et al 1998, chapter 5.7). Only applicable when `inserts = "map"`.
`threshold`	the maximum proportion of gaps for an alignment column to be considered for a match state in the PHMM (defaults to 0.5). Only applicable when `inserts = "threshold"`. Note that the maximum *a posteriori* method works poorly for alignments with few sequences, so the 'threshold' method is automatically used when the number of sequences is less than 5.
`deltaLL`	numeric, the maximum change in log likelihood between EM iterations before the cycling procedure is terminated (signifying model convergence). Defaults to 1E-07. Only applicable if `method = "BaumWelch"`.
`DI`	logical indicating whether delete-insert transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`ID`	logical indicating whether insert-delete transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors. Furthermore, the default setting `residues = NULL` will not detect rare residues that are not present in the sequences, and thus will not assign them emission probabilities in the model. Specifying the residue alphabet is therefore recommended unless x is a "DNAbin" or "AAbin" object.
`gap`	the character used to represent gaps in the alignment matrix. Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "-" otherwise.
`pseudocounts`	character string, either "background", Laplace" or "none". Used to account for the possible absence of certain transition and/or emission types in the input sequences. If `pseudocounts = "background"` (default), pseudocounts are calculated from the background transition and emission frequencies in the sequences. If `pseudocounts = "Laplace"` one of each possible transition and emission type is added to the transition and emission counts. If `pseudocounts = "none"` no pseudocounts are added (not generally recommended, since low frequency transition/emission types may be excluded from the model). Alternatively this argument can be a two-element list containing a matrix of transition pseudocounts as its first element and a matrix of emission pseudocounts as its second.
`qa`	an optional named 9-element vector of background transition probabilities with `dimnames(qa) = c("DD", "DM", "DI", "MD", "MM","MI", "ID", "IM", "II")`, where M, I and D represent match, insert and delete states, respectively. If `NULL`, background transition probabilities are estimated from the sequences.
`qe`	an optional named vector of background emission probabilities the same length as the residue alphabet (i.e. 4 for nucleotides and 20 for amino acids) and with corresponding names (i.e. `c("A", "T","G", "C")` for DNA). If `qe = NULL`, background emission probabilities are automatically derived from the sequences.
`cores`	integer giving the number of CPUs to parallelize the operation over. Defaults to 1, and reverts to 1 if x is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection

	at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be aligned, due to the extra time required to initialize the cluster.
`quiet`	logical indicating whether feedback should be printed to the console.

Details

This function builds a multiple sequence alignment using profile hidden Markov models. The default behaviour is to select the longest sequence in the set that had the lowest sequence weight, derive a profile HMM from the single sequence, and iteratively train the model using the entire sequence set. Training can be achieved using either the Baum Welch or Viterbi training algorithm, with the latter being significantly faster, particularly when multi-threading is used. Once the model parameters have converged (Baum Welch) or no variation is seen in the sequential alignments (Viterbi training), the sequences are aligned to the profile HMM to produce the alignment matrix. The preceeding steps can be omitted if a pre-trained profile HMM is passed to the function via the "model" argument.

If `progressive = TRUE` the function alternatively uses a progressive alignment procedure similar to the Clustal Omega algorithm (Sievers et al 2011). The involves an initial progressive multiple sequence alignment via a guide tree, followed by the derivation of a profile hidden Markov model from the alignment, an iterative model refinement step, and finally the alignment of the sequences back to the model as above.

If only two sequences are provided, a standard pairwise alignment is carried out using the Needleman-Wunch or Smith-Waterman algorithm.

Value

a matrix of aligned sequences, with the same mode and class as the input sequence list.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, Li W, Lopez R, McWilliam H, Remmert M, Soding J, Thompson JD, Higgins DG (2011) Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, **7**, 539.

See Also

`unalign`

Examples

```
## Protein pairwise alignment example from Durbin et al (1998) chapter 2.
x <- c("H", "E", "A", "G", "A", "W", "G", "H", "E", "E")
y <- c("P", "A", "W", "H", "E", "A", "E")
sequences <- list(x = x, y = y)
glo <- align(sequences, type = "global")
sem <- align(sequences, type = "semiglobal")
loc <- align(sequences, type = "local")
glo
sem
loc

## Deconstruct the woodmouse alignment and re-align
library(ape)
data(woodmouse)
tmp <- unalign(woodmouse)
x <- align(tmp, windowspace = "WilburLipman")
```

---

aphid — *The* **aphid** *package for analysis with profile hidden Markov models.*

Description

aphid is an R package for the development and application of hidden Markov models and profile HMMs for biological sequence analysis. Functions are included for multiple and pairwise sequence alignment, model construction and parameter optimization, calculation of conditional probabilities (using the forward, backward and Viterbi algorithms), tree-based sequence weighting, sequence simulation, and file import/export compatible with the HMMER software package. The package has a wide variety of uses including database searching, gene-finding and annotation, phylogenetic analysis and sequence classification.

Details

The **aphid** package is based on the algorithms outlined in the book 'Biological sequence analysis: probabilistic models of proteins and nucleic acids' by Richard Durbin, Sean Eddy, Anders Krogh and Graeme Mitchison. This book is highly recommended for those wishing to develop a better understanding of HMMs and PHMMs, regardless of prior experience. Many of the examples in the function help pages are taken directly from the book, so that readers can learn to use the package as they work through the chapters.

There are also excellent rescources available for those wishing to use profile hidden Markov models outside of the R environment. The **aphid** package maintains compatibility with the HMMER software suite through the file input and output functions `readPHMM` and `writePHMM`. Those interested are further encouraged to check out the SAM software package, which also features a comprehensive suite of functions and tutorials.

The **aphid** package is designed to work in conjunction with the "DNAbin" and "AAbin" object types produced by the `ape` package (Paradis et al 2004, 2012). This is an essential piece of software for those using R for biological sequence analysis, and provides a binary coding format for nucleotides and amino acids that maximizes memory and speed efficiency. While **aphid** also works with standard character vectors and matrices, it may not recognize the DNA and amino acid amibguity codes and therefore is not guaranteed to treat them appropriately.

To maximize speed, the low-level dynamic programming functions such as `Viterbi`, `forward` and `backward` are written in C++ with the help of the `Rcpp` package (Eddelbuettel & Francois 2011). Note that R versions of these functions are also maintained for the purposes of debugging, experimentation and code interpretation.

Classes

The **aphid** package creates two primary object classes, `"HMM"` (hidden Markov models) and `"PHMM"` (profile hidden Markov models) with the functions `deriveHMM` and `derivePHMM`, respectively. These objects are lists consisting of emission and transition probability matrices (denoted E and A), vectors of non-position-specific background emission and transition probabilies (denoted qe and qa) and other model metadata. Objects of class `"DPA"` (dynammic programming array) are also generated by the Viterbi and forward/backward functions. These are primarily created for succinct console printing.

Functions

A breif description of the primary **aphid** functions are provided with links to their help pages below.

File import and export

- `readPHMM` parses a HMMER text file into R and creates an object of class `"PHMM"`
- `writePHMM` writes a `"PHMM"` object to a text file in HMMER v3 format

Visualization

- `plot.HMM` plots a `"PHMM"` object as a cyclic directed graph
- `plot.PHMM` plots a `"PHMM"` object as a directed graph with sequential modules consisting of match, insert and delete states

Model building and training

- `deriveHMM` builds a `"HMM"` object from a list of training sequences
- `derivePHMM` builds a `"PHMM"` object from a multiple sequence alignment or a list of non-aligned sequences
- `map` optimizes profile hidden Markov model construction using the maximum *a posteriori* algorithm
- `train` optimizes the parameters of a `"HMM"` or `"PHMM"` object using a list of training sequences

Sequence alignment and weighting

- `align` performs a multiple sequence alignment
- `weight` assigns weights to sequences

Conditional probabilities

- `Viterbi` finds the optimal path of a sequence through a HMM or PHMM, and returns its log odds or probability given the model
- `forward` finds the full probability of a sequence given a HMM or PHMM using the forward algorithm
- `backward` finds the full probability of a sequence given a HMM or PHMM using the backward algorithm
- `posterior` finds the position-specific posterior probability of a sequence given a HMM or PHMM

Sequence simulation

- `generate.HMM` simulates a random sequence from an HMM
- `generate.PHMM` simulates a random sequence from a PHMM

Datasets

- `substitution` a collection of DNA and amino acid substitution matrices from NCBI including the PAM, BLOSUM, GONNET, DAYHOFF and NUC matrices
- `casino` data from the dishonest casino example of Durbin et al (1998) chapter 3.2
- `globins` Small globin alignment data from Durbin et al (1998) Figure 5.3

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Eddelbuettel D, Francois R (2011) Rcpp: seamless R and C++ integration. *Journal of Statistical Software* **40**, 1-18.

Finn RD, Clements J & Eddy SR (2011) HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*. **39**, W29-W37. `http://hmmer.org/`.

HMMER: biosequence analysis using profile hidden Markov models. `http://www.hmmer.org`.

NCBI index of substitution matrices. `ftp://ftp.ncbi.nih.gov/blast/matrices/`.

Paradis E, Claude J, Strimmer K, (2004) APE: analyses of phylogenetics and evolution in R language. *Bioinformatics* **20**, 289-290.

Paradis E (2012) Analysis of Phylogenetics and Evolution with R (Second Edition). Springer, New York.

See Also

Useful links:

- https://github.com/shaunpwilkinson/aphid
- Report bugs at https://github.com/shaunpwilkinson/aphid/issues

---

`backward` *The backward algorithm.*

Description

This function calculates the full (log) probability or odds of a sequence given a hidden Markov model or profile HMM using the backward dynamic programming algorithm.

Usage

```
backward(x, y, ...)

## S3 method for class 'PHMM'
backward(
  x,
  y,
  qe = NULL,
  logspace = "autodetect",
  odds = TRUE,
  windowspace = "all",
  DI = FALSE,
  ID = FALSE,
  cpp = TRUE,
  ...
)

## S3 method for class 'HMM'
backward(x, y, logspace = "autodetect", cpp = TRUE, ...)
```

Arguments

`x`	an object of class `PHMM` or `HMM`.
`y`	a vector of mode "character" or "raw" (a "DNAbin" or "AAbin" object) representing a single sequence hypothetically emitted by the model in `x`.
`...`	additional arguments to be passed between methods.
`qe`	an optional named vector of background residue frequencies (only applicable if x is a PHMM). If `qe = NULL` the function looks for a qe vector as an attribute of the PHMM. If these are not available equal background residue frequencies are assumed.

`logspace`	logical indicating whether the emission and transition probabilities of `x` are logged. If `logspace = "autodetect"` (default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`odds`	logical, indicates whether the returned scores should be odds ratios (TRUE) or full logged probabilities (FALSE).
`windowspace`	a two-element integer vector providing the search space for dynamic programming (see Wilbur & Lipman 1983 for details). The first element should be negative, and represent the lowermost diagonal of the dynammic programming array, and the second element should be positive, representing the leftmost diagonal. Alternatively, if the the character string "all" is passed (the default setting) the entire dynamic programming array will be computed.
`DI`	logical indicating whether delete-insert transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`ID`	logical indicating whether insert-delete transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The FALSE option is primarily retained for bug fixing and experimentation.

Details

This function is a wrapper for a compiled C++ function that recursively fills a dynamic programming matrix with logged probabilities, and calculates the full (logged) probability of a sequence given a HMM or PHMM. For a thorough explanation of the backward, forward and Viterbi algorithms, see Durbin et al (1998) chapters 3.2 (HMMs) and 5.4 (PHMMs).

Value

an object of class `"DPA"`, which is a list containing the score and dynamic programming array.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Wilbur WJ, Lipman DJ (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA*, **80**, 726-730.

See Also

`forward`, `Viterbi`.

Examples

```
## Backward algorithm for standard HMMs:
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino HMM")
data(casino)
backward(x, casino)
##
## Backward algorithm for profile HMMs:
## Small globin alignment data from Durbin et al (1998) Figure 5.3
data(globins)
### Derive a profile hidden Markov model from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
plot(globins.PHMM, main = "Profile hidden Markov model for globins")
### Simulate a random sequence from the model
suppressWarnings(RNGversion("3.5.0"))
set.seed(999)
simulation <- generate(globins.PHMM, size = 20)
simulation ## "F" "S" "A" "N" "N" "D" "W" "E"
### Calculate the full (log) probability of the sequence given the model
x <- backward(globins.PHMM, simulation, odds = FALSE)
x # -23.0586
### Show dynammic programming array
x$array
```

---

casino *Dishonest casino.*

Description

The 'dishonest casino' example from Durbin et al (1998) chapter 3.2.

Usage

```
casino
```

Format

A named character vector showing the result of 300 rolls of a dice that switches from "Fair" to "Loaded" with a probability of 0.05 and back to "Fair" with a probability of 0.1. In the Fair state each outcome from 1 to 6 has an equal probability of occurring, while in the Loaded state the probability of rolling a 6 increases to 0.5 (with the remaining five probabilities reduced to 0.1). The elements of the vector are the outcomes of the 300 rolls ("1", "2", "3", "4", "5", or "6") and the "names" attribute represents the underlying Markov states ("Fair" or "Loaded").

Source

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

`deriveHMM` *Derive a standard hidden Markov model from a set of sequences.*

Description

`deriveHMM` calculates the maximum likelihood hidden Markov model from a list of training sequences, each a vector of residues named according the state from which they were emitted.

Usage

```
deriveHMM(
  x,
  seqweights = NULL,
  residues = NULL,
  states = NULL,
  modelend = FALSE,
  pseudocounts = "background",
  logspace = TRUE
)
```

Arguments

`x`	a list of named character vectors representing emissions from the model. The 'names' attribute should represent the hidden state from which each residue was emitted. "DNAbin" and "AAbin" list objects are also supported for modeling DNA or amino acid sequences.
`seqweights`	either NULL (all sequences are given weights of 1) or a numeric vector the same length as `x` representing the sequence weights used to derive the model.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors. Furthermore, the default setting `residues = NULL` will not detect rare residues that are not present in the sequences, and thus will not assign them emission probabilities in the model. Specifying the residue alphabet is therefore recommended unless x is a "DNAbin" or "AAbin" object.

`states`	either NULL (default; the unique Markov states are automatically detected from the 'names' attributes of the input sequences), or a case sensitive character vector specifying the unique Markov states (or a superset of the unique states) to appear in the model. The latter option is recommended since it saves computation time and ensures that all valid Markov states appear in the model, regardless of their possible absence from the training dataset.
`modelend`	logical indicating whether transition probabilites to the end state of the standard hidden Markov model should be modeled (if applicable). Defaults to FALSE.
`pseudocounts`	character string, either "background", Laplace" or "none". Used to account for the possible absence of certain transition and/or emission types in the input sequences. If `pseudocounts = "background"` (default), pseudocounts are calculated from the background transition and emission frequencies in the training dataset. If `pseudocounts = "Laplace"` one of each possible transition and emission type is added to the training dataset (default). If `pseudocounts = "none"` no pseudocounts are added (not usually recommended, since low frequency transition/emission types may be excluded from the model). Alternatively this argument can be a two-element list containing a matrix of transition pseudocounts as its first element and a matrix of emission pseudocounts as its second. If this option is selected, both matrices must have row and column names corresponding with the residues (column names of emission matrix) and states (row and column names of the transition matrix and row names of the emission matrix). For downstream applications the first row and column of the transition matrix should be named "Begin".
`logspace`	logical indicating whether the emission and transition probabilities in the returned model should be logged. Defaults to TRUE.

Details

This function creates a standard hidden Markov model (object class: `"HMM"`) using the method described in Durbin et al (1998) chapter 3.3. It assumes the state sequence is known (as opposed to the `train.HMM` function, which is used when the state sequence is unknown) and provided as the names attribute(s) of the input sequences. The output object is a simple list with elements "A" (transition probability matrix) and "E" (emission probability matrix), and the "class" attribute "HMM". The emission matrix has the same number of rows as the number of states, and the same number of columns as the number of unique symbols that can be emitted (i.e. the residue alphabet). The number of rows and columns in the transition probability matrix should be one more the number of states, to include the silent "Begin" state in the first row and column. Despite its name, this state is also used when modeling transitions to the (silent) end state, which are entered in the first column.

Value

an object of class `"HMM"`.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

derivePHMM

Examples

```
data(casino)
deriveHMM(list(casino))
```

---

derivePHMM *Derive a profile hidden Markov model from sequences.*

Description

derivePHMM generates a profile HMM from a given multiple sequence alignment or a list of unaligned sequences.

Usage

```
derivePHMM(x, ...)

## S3 method for class 'DNAbin'
derivePHMM(
  x,
  seqweights = "Henikoff",
  wfactor = 1,
  k = 5,
  residues = NULL,
  gap = "-",
  endchar = "?",
  pseudocounts = "background",
  logspace = TRUE,
  qa = NULL,
  qe = NULL,
  maxsize = NULL,
  inserts = "map",
  threshold = 0.5,
  lambda = 0,
  DI = FALSE,
  ID = FALSE,
  omit.endgaps = FALSE,
  name = NULL,
  description = NULL,
  compo = FALSE,
  consensus = FALSE,
  alignment = FALSE,
  progressive = FALSE,
  seeds = NULL,
  refine = "Viterbi",
  maxiter = 100,
  deltaLL = 1e-07,
  cpp = TRUE,
  cores = 1,
  quiet = FALSE,
  ...
)

## S3 method for class 'AAbin'
derivePHMM(
  x,
  seqweights = "Henikoff",
  wfactor = 1,
  k = 5,
  residues = NULL,
  gap = "-",
  endchar = "?",
  pseudocounts = "background",
  logspace = TRUE,
  qa = NULL,
  qe = NULL,
  maxsize = NULL,
  inserts = "map",
  threshold = 0.5,
  lambda = 0,
  DI = FALSE,
  ID = FALSE,
  omit.endgaps = FALSE,
  name = NULL,
  description = NULL,
  compo = FALSE,
  consensus = FALSE,
  alignment = FALSE,
  progressive = FALSE,
  seeds = NULL,
  refine = "Viterbi",
  maxiter = 100,
  deltaLL = 1e-07,
  cpp = TRUE,
  cores = 1,
  quiet = FALSE,
  ...
```

```
)

## S3 method for class 'list'
derivePHMM(
  x,
  progressive = FALSE,
  seeds = NULL,
  refine = "Viterbi",
  maxiter = 100,
  deltaLL = 1e-07,
  seqweights = "Henikoff",
  wfactor = 1,
  k = 5,
  residues = NULL,
  gap = "-",
  pseudocounts = "background",
  logspace = TRUE,
  qa = NULL,
  qe = NULL,
  maxsize = NULL,
  inserts = "map",
  lambda = 0,
  DI = FALSE,
  ID = FALSE,
  threshold = 0.5,
  omit.endgaps = FALSE,
  name = NULL,
  description = NULL,
  compo = FALSE,
  consensus = FALSE,
  alignment = FALSE,
  cpp = TRUE,
  cores = 1,
  quiet = FALSE,
  ...
)

## Default S3 method:
derivePHMM(
  x,
  seqweights = "Henikoff",
  wfactor = 1,
  k = 5,
  residues = NULL,
  gap = "-",
  endchar = "?",
  pseudocounts = "background",
  logspace = TRUE,
  qa = NULL,
  qe = NULL,
  maxsize = NULL,
  inserts = "map",
  lambda = 0,
  threshold = 0.5,
  DI = FALSE,
  ID = FALSE,
  omit.endgaps = FALSE,
  name = NULL,
  description = NULL,
  compo = FALSE,
  consensus = FALSE,
  alignment = FALSE,
  cpp = TRUE,
  quiet = FALSE,
  ...
)
```

Arguments

`x`	a matrix of aligned sequences or a list of unaligned sequences. Accepted modes are "character" and "raw" (for "DNAbin" and "AAbin" objects).
`...`	aditional arguments to be passed to `"Viterbi"` (if `refine = "Viterbi"`) or `"forward"` (if `refine = "BaumWelch"`).
`seqweights`	either NULL (all sequences are given weights of 1), a numeric vector the same length as `x` representing the sequence weights used to derive the model, or a character string giving the method to derive the weights from the sequences (see `weight`).
`wfactor`	numeric. The factor to multiply the sequence weights by. Defaults to 1.
`k`	integer representing the k-mer size to be used in tree-based sequence weighting (if applicable). Defaults to 5. Note that higher values of k may be slow to compute and use excessive memory due to the large numbers of calculations required.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors. Furthermore, the default setting `residues = NULL` will not detect rare residues that are not present in the sequences, and thus will not assign them emission probabilities in the model. Specifying the residue alphabet is therefore recommended unless x is a "DNAbin" or "AAbin" object.
`gap`	the character used to represent gaps in the alignment matrix. Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "-" otherwise.
`endchar`	the character used to represent unknown residues in the alignment matrix (if applicable). Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "?" otherwise.

`pseudocounts`	character string, either "background", Laplace" or "none". Used to account for the possible absence of certain transition and/or emission types in the input sequences. If `pseudocounts = "background"` (default), pseudocounts are calculated from the background transition and emission frequencies in the sequences. If `pseudocounts = "Laplace"` one of each possible transition and emission type is added to the transition and emission counts. If `pseudocounts = "none"` no pseudocounts are added (not generally recommended, since low frequency transition/emission types may be excluded from the model). Alternatively this argument can be a two-element list containing a matrix of transition pseudocounts as its first element and a matrix of emission pseudocounts as its second.
`logspace`	logical indicating whether the emission and transition probabilities in the returned model should be logged. Defaults to TRUE.
`qa`	an optional named 9-element vector of background transition probabilities with `dimnames(qa) = c("DD", "DM", "DI", "MD", "MM","MI", "ID", "IM", "II")`, where M, I and D represent match, insert and delete states, respectively. If `NULL`, background transition probabilities are estimated from the sequences.
`qe`	an optional named vector of background emission probabilities the same length as the residue alphabet (i.e. 4 for nucleotides and 20 for amino acids) and with corresponding names (i.e. `c("A", "T","G", "C")` for DNA). If `qe = NULL`, background emission probabilities are automatically derived from the sequences.
`maxsize`	integer giving the upper bound on the number of modules in the PHMM. If NULL (default) no maximum size is enforced.
`inserts`	character string giving the model construction method by which alignment columns are marked as either match or insert states. Accepted methods include `"threshold"` (only columns with fewer than a specified proportion of gaps form match states in the model), `"map"` (default; match and insert columns are found using the maximum *a posteriori* method outlined in Durbin et al (1998) chapter 5.7), `"inherited"` (match and insert columns are inherited from the input alignment), and `"none"` (all columns are assigned match states in the model). Alternatively, insert columns can be specified manually by providing a logical vector the same length as the number of columns in the alignment, with `TRUE` for insert columns and `FALSE` for match states.
`threshold`	the maximum proportion of gaps for an alignment column to be considered for a match state in the PHMM (defaults to 0.5). Only applicable when `inserts = "threshold"`. Note that the maximum *a posteriori* method works poorly for alignments with few sequences, so the 'threshold' method is automatically used when the number of sequences is less than 5.
`lambda`	penalty parameter used to favour models with fewer match states. Equivalent to the log of the prior probability of marking each column (Durbin et al 1998, chapter 5.7). Only applicable when `inserts = "map"`.
`DI`	logical indicating whether delete-insert transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`ID`	logical indicating whether insert-delete transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`omit.endgaps`	logical. Should gap characters at each end of the sequences be ignored when deriving the transition probabilities of the model? Defaults to FALSE. Set to

	TRUE if `x` is not a strict global alignment (i.e. if the alignment contains partial sequences with missing sections represented with gap characters).
`name`	an optional character string. The name of the new profile hidden Markov model.
`description`	an optional character string. The description of the new profile hidden Markov model.
`compo`	logical indicating whether the average emission probabilities of the model modules should be returned with the PHMM object.
`consensus`	placeholder. Consensus sequences will be available in a future version.
`alignment`	logical indicating whether the alignment used to derive the final model (if applicable) should be included as an element of the returned PHMM object. Defaults to FALSE.
`progressive`	logical indicating whether the alignment used to derive the initial model parameters should be built progressively (assuming input is a list of unaligned sequences, ignored otherwise). Defaults to FALSE, in which case the longest sequence or sequences are used (faster, but possibly less accurate).
`seeds`	optional integer vector indicating which sequences should be used as seeds for building the guide tree for the progressive alignment (assuming input is a list of unaligned sequences, and `progressive = TRUE`, ignored otherwise). Defaults to NULL, in which a set of log(n, 2)^2 non-identical sequences are chosen from the list of sequences by k-means clustering.
`refine`	the method used to iteratively refine the model parameters following the initial progressive alignment and model derivation step. Current supported options are `"Viterbi"` (Viterbi training; the default option), `"BaumWelch"` (a modified version of the Expectation-Maximization algorithm), and "none" (skips the model refinement step).
`maxiter`	the maximum number of EM iterations or Viterbi training iterations to carry out before the cycling process is terminated and the partially trained model is returned. Defaults to 100.
`deltaLL`	numeric, the maximum change in log likelihood between EM iterations before the cycling procedure is terminated (signifying model convergence). Defaults to 1E-07. Only applicable if `method = "BaumWelch"`.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The FALSE option is primarily retained for bug fixing and experimentation.
`cores`	integer giving the number of CPUs to parallelize the operation over. Defaults to 1, and reverts to 1 if x is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be aligned, due to the extra time required to initialize the cluster.
`quiet`	logical indicating whether feedback should be printed to the console.

Details

This function performs a similar operation to the `hmmbuild` function in the HMMER package, and the `modelfromalign` and `buildmodel` functions in the SAM package. If the primary input argument is an alignment, the function creates a profile hidden Markov model (object class:`"PHMM"`) using the method described in Durbin et al (1998) chapter 5.3. Alternatively, if a list of non-aligned sequences is passed, the sequences are first aligned using the `align` function before being used to derive the model.

The function outputs an object of class `"PHMM"`, which is a list consisting of emission and transition probability matrices (elements named "E" and "A"), vectors of non-position-specific background emission and transition probabilities ("qe" and "qa", respectively) and other model metadata including "name", "description", "size" (the number of modules in the model), and "alphabet" (the set of symbols/residues emitted by the model).

Value

an object of class `"PHMM"`

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

`deriveHMM`, `map`

Examples

```
## Small globin alignment data from Durbin et al (1998) Figure 5.3
data(globins)
## derive a profile hidden Markov model from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
plot(globins.PHMM, main = "Profile HMM for small globin alignment")
##
## derive a profle HMM from the woodmouse dataset in the
## ape package and plot the first 5 modules
library(ape)
data(woodmouse)
woodmouse.PHMM <- derivePHMM(woodmouse)
plot(woodmouse.PHMM, from = 0, to = 5, main = "Partial woodmouse profile HMM")
```

forward *The forward algorithm.*

Description

This function calculates the full (log) probability or odds of a sequence given a hidden Markov model or profile HMM using the forward dynamic programming algorithm.

Usage

```
forward(x, y, ...)

## S3 method for class 'PHMM'
forward(
  x,
  y,
  qe = NULL,
  logspace = "autodetect",
  odds = TRUE,
  windowspace = "all",
  DI = FALSE,
  ID = FALSE,
  cpp = TRUE,
  ...
)

## S3 method for class 'HMM'
forward(x, y, logspace = "autodetect", cpp = TRUE, ...)
```

Arguments

`x`	an object of class `PHMM` or `HMM`.
`y`	a vector of mode "character" or "raw" (a "DNAbin" or "AAbin" object) representing a single sequence hypothetically emitted by the model in `x`.
`...`	additional arguments to be passed between methods.
`qe`	an optional named vector of background residue frequencies (only applicable if x is a PHMM). If `qe = NULL` the function looks for a qe vector as an attribute of the PHMM. If these are not available equal background residue frequencies are assumed.
`logspace`	logical indicating whether the emission and transition probabilities of `x` are logged. If `logspace = "autodetect"` (default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`odds`	logical, indicates whether the returned scores should be odds ratios (TRUE) or full logged probabilities (FALSE).

`windowspace`	a two-element integer vector providing the search space for dynamic programming (see Wilbur & Lipman 1983 for details). The first element should be negative, and represent the lowermost diagonal of the dynammic programming array, and the second element should be positive, representing the leftmost diagonal. Alternatively, if the the character string "all" is passed (the default setting) the entire dynamic programming array will be computed.
`DI`	logical indicating whether delete-insert transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`ID`	logical indicating whether insert-delete transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The FALSE option is primarily retained for bug fixing and experimentation.

Details

This function is a wrapper for a compiled C++ function that recursively fills a dynamic programming matrix with logged probabilities, and calculates the full (logged) probability of a sequence given a HMM or PHMM.

For a thorough explanation of the backward, forward and Viterbi algorithms, see Durbin et al (1998) chapters 3.2 (HMMs) and 5.4 (PHMMs).

Value

an object of class `"DPA"`, which is a list containing the score and dynamic programming array.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Wilbur WJ, Lipman DJ (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA*, **80**, 726-730.

See Also

`backward`, `Viterbi`.

Examples

```
## Forward algorithm for standard HMMs:
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
```

```
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino HMM")
### Find full probability of the sequence given the model
data(casino)
forward(x, casino)
###
## Forward algorithm for profile HMMs:
## Small globin alignment data from Durbin et al (1998) Figure 5.3
data(globins)
### Derive a profile HMM from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
plot(globins.PHMM, main = "Profile hidden Markov model for globins")
### Simulate a random sequence from the model
suppressWarnings(RNGversion("3.5.0"))
set.seed(999)
simulation <- generate(globins.PHMM, size = 20)
simulation ## "F" "S" "A" "N" "N" "D" "W" "E"
### Calculate the full (log) probability of the sequence given the model
x <- forward(globins.PHMM, simulation, odds = FALSE)
x # -23.0586
### Show the dynammic programming array
x$array
```

`generate` *Generate random sequences from a model.*

Description

The `generate` function outputs a random sequence from a HMM or PHMM.

Usage

```
generate(x, size, ...)

## S3 method for class 'HMM'
generate(x, size, logspace = "autodetect", random = TRUE, ...)

## S3 method for class 'PHMM'
generate(
  x,
  size,
  logspace = "autodetect",
  gap = "-",
  random = TRUE,
  DNA = FALSE,
  AA = FALSE,
  ...
)
```

Arguments

`x`	an object of class `'HMM'` or `'PHMM'`.
`size`	a non-negative integer representing the length of the output sequence if `x` is a `"HMM"` object with zero probability of transitioning to the begin/end state, or the maximum length of the output sequence otherwise (this acts as a safeguard against overflow).
`...`	additional arguments to be passed between methods.
`logspace`	logical indicating whether the emission and transition probabilities of `x` are logged. If `logspace = "autodetect"` (the default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`random`	logical indicating whether residues should be emitted randomly with probabilities defined by the emission probabilities in the model (TRUE; default), or deterministically, whereby each residue is emitted and each transition taken based on the maximum emission/transition probability in the current state.
`gap`	the character used to represent gaps (delete states) in the output sequence (only applicable for `PHMM` objects).
`DNA`	logical indicating whether the returned sequence should be a `"DNAbin"` object. Only applicable if the matrix of emission probabilities in the model has four residues corresponding to the nucleotide alphabet (A, T, G, and C).
`AA`	logical indicating whether the returned sequence should be a `"AAbin"` object. Only applicable if the matrix of emission probabilities in the model has 20 residues corresponding to the amino acid alphabet.

Details

This simple function generates a single sequence from a HMM or profile HMM by recursively simulating a path through the model. The function is fairly slow in its current state, but a faster C++ function may be made available in a future version depending on demand.

Value

a named vector giving the sequence of residues emitted by the model, with the "names" attribute representing the hidden states.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Examples

```
## Generate a random sequence from a standard HMM
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino HMM")
### Generate a random sequence from the model
generate(x, size = 300)
##
## Generate a random sequence from a profile HMM:
## Small globin alignment data from Durbin et al (1998) Figure 5.3
data(globins)
### Derive a profile hidden Markov model from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
plot(globins.PHMM, main = "Profile hidden Markov model for globins")
### Simulate a random sequence from the model
suppressWarnings(RNGversion("3.5.0"))
set.seed(999)
simulation <- generate(globins.PHMM, size = 20)
simulation ## "F" "S" "A" "N" "N" "D" "W" "E"
### Names attribute indicates that all residues came from "match" states
```

`globins` *Globin protein alignment.*

Description

The small globin protein alignment from figure 5.3 of Durbin et al (1998).

Usage

```
globins
```

Format

a 7 x 10 character matrix with ten columns of a multiple alignment of globin amino acid sequences from Durbin et al (1998) chapter 5.3.

Source

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

`logsum` *Sum of logged probabilities.*

Description

`"logsum"` takes a vector of logged probabilities (neagtive values) and returns its sum.

Usage

```
logsum(x)
```

Arguments

`x`	a numeric vector of logged probabilities.

Details

This is a simple compiled function that exponentiates the values in the input vector, finds their sum, and returns the log of that value.

Value

returns a single numeric value representing the logged sum of the values in the input vector.

Author(s)

Shaun Wilkinson

`map` *Optimized profile HMM construction.*

Description

Assigns match and insert states to alignment columns using the maximum *a posteriori* algorithm outlined in Durbin et al (1998) chapter 5.7.

Usage

```
map(
  x,
  seqweights = NULL,
  residues = NULL,
  gap = "-",
  endchar = "?",
  pseudocounts = "background",
  lambda = 0,
  qa = NULL,
  qe = NULL,
  cpp = TRUE
)
```

Arguments

`x`	a matrix of aligned sequences. Accepted modes are "character" and "raw" (the latter being used for "DNAbin" and "AAbin" objects).
`seqweights`	either NULL (default; all sequences are given weights of 1) or a numeric vector the same length as `x` representing the sequence weights used to derive the model.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors. Furthermore, the default setting `residues = NULL` will not detect rare residues that are not present in the sequences, and thus will not assign them emission probabilities in the model. Specifying the residue alphabet is therefore recommended unless x is a "DNAbin" or "AAbin" object.
`gap`	the character used to represent gaps in the alignment matrix (if applicable). Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "-" otherwise.
`endchar`	the character used to represent unknown residues in the alignment matrix (if applicable). Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "?" otherwise.
`pseudocounts`	character string, either "background", Laplace" or "none". Used to account for the possible absence of certain transition and/or emission types in the input sequences. If `pseudocounts = "background"` (default), pseudocounts are calculated from the background transition and emission frequencies in the sequences. If `pseudocounts = "Laplace"` one of each possible transition and emission type is added to the transition and emission counts. If `pseudocounts = "none"` no pseudocounts are added (not generally recommended, since low frequency transition/emission types may be excluded from the model). Alternatively this argument can be a two-element list containing a matrix of transition pseudocounts as its first element and a matrix of emission pseudocounts as its second.
`lambda`	penalty parameter used to favour models with fewer match states. Equivalent to the log of the prior probability of marking each column (Durbin et al 1998, chapter 5.7).

`qa`	an optional named 9-element vector of background transition probabilities with `dimnames(qa) = c("DD", "DM", "DI", "MD", "MM","MI", "ID", "IM", "II")`, where M, I and D represent match, insert and delete states, respectively. If `NULL`, background transition probabilities are estimated from the sequences.
`qe`	an optional named vector of background emission probabilities the same length as the residue alphabet (i.e. 4 for nucleotides and 20 for amino acids) and with corresponding names (i.e. `c("A", "T","G", "C")` for DNA). If `qe = NULL`, background emission probabilities are automatically derived from the sequences.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The FALSE option is primarily retained for bug fixing and experimentation.

Details

see Durbin et al (1998) chapter 5.7 for details of the maximum *a posteriori* algorithm for optial match and insert state assignment.

Value

a logical vector with length = ncol(x) indicating the columns to be assigned as match states (`TRUE`) and those assigned as inserts (`FALSE`).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

`derivePHMM`

Examples

```
## Maximum a posteriori assignment of match states to the small
## alignment example in Figure 5.3, Durbin et al (1998)
data(globins)
map(globins)
```

plot.HMM *Plot standard hidden Markov models.*

Description

`plot.HMM` provides a visual representation of a standard hidden Markov model.

Usage

```
## S3 method for class 'HMM'
plot(x, just = "center", arrexp = 1, textexp = 1, begin = FALSE, ...)
```

Arguments

`x`	an object of class `"HMM"`.
`just`	a character string giving the justfication of the plot relative to the device. Accepted values are "left", "center" and "right".
`arrexp`	the expansion factor to be applied to the arrows in the plot.
`textexp`	the expansion factor to be applied to the text in the plot.
`begin`	logical indicating whether the begin/end state should be plotted. Defaults to FALSE.
`...`	additional arguments to be passed to `plot`.

Details

`"plot.HMM"` Plots a `"HMM"` object as a directed graph. States (rectangles) are interconnected by directed lines with line-weights proportional to the transition probabilities between the states.

Value

NULL (invisibly).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

`plot.PHMM`

Examples

```
## the dishonest casino example from Durbin et al (1998)
states <- c("Begin", "Fair", "Loaded")
residues = paste(1:6)
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino hidden Markov model")
```

---

plot.PHMM *Plot profile hidden Markov models.*

Description

plot.PHMM provides a visual representation of a profile hidden Markov model.

Usage

```
## S3 method for class 'PHMM'
plot(
  x,
  from = "start",
  to = "end",
  just = "center",
  arrexp = 1,
  textexp = 1,
  ...
)
```

Arguments

x	an object of class "PHMM".
from	an integer giving the module number to start the plot sequence from. Also accepts the chracter string "start" (module 0; default).
to	an integer giving the module number to terminate the plot sequence. Also accepts the chracter string "end" (default).
just	a character string giving the justfication of the plot relative to the device. Accepted values are "left", "center" and "right".
arrexp	the expansion factor to be applied to the arrows in the plot.
textexp	the expansion factor to be applied to the text in the plot.
...	additional arguments to be passed to plot.

Details

"plot.PHMM" Plots a "PHMM" object as a directed graph with sequential modules consisting of squares, diamonds and circles representing match, insert and delete states, respectively. Modules are interconnected by directed lines with line-weights proportional to the transition probabilities between the states. Since the plotted models are generally much longer than they are high, it is usually better to output the plot to a PDF file as demonstrated in the example below.

Value

NULL (invisibly).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

plot.HMM

Examples

```
## Small globin alignment example from Durbin et al (1998) Figure 5.3
data(globins)
## derive a profile hidden Markov model from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
## plot the PHMM
plot(globins.PHMM, main = "Profile hidden Markov model for globins")
##
## derive a profile hidden Markov model from the woodmouse dataset in the
## ape package
library(ape)
data(woodmouse)
woodmouse.PHMM <- derivePHMM(woodmouse)
## plot partial model to viewer device
plot(woodmouse.PHMM, from = 0, to = 5)
## plot the entire model to a PDF in the current working directory

tmpf <- tempfile(fileext = ".pdf")
nr <- ceiling((woodmouse.PHMM$size + 2)/10)
pdf(file = tmpf, width = 8.27, height = nr * 2)
par(mfrow = c(nr, 1), mar = c(0, 0, 0, 0) + 0.1)
from <- 0
to <- 10
for(i in 1:nr){
  plot(woodmouse.PHMM, from = from, to = to, just = "left")
  from <- from + 10
```

```
    to <- min(to + 10, woodmouse.PHMM$size + 1)
  }
  dev.off()
```

`posterior` *Posterior decoding.*

Description

Calculate the posterior probability of a sequence given a model.

Usage

```
posterior(x, y, ...)

## S3 method for class 'HMM'
posterior(x, y, logspace = "autodetect", cpp = TRUE, ...)

## S3 method for class 'PHMM'
posterior(x, y, logspace = "autodetect", cpp = TRUE, ...)
```

Arguments

`x`	an object of class `'HMM'` or `'PHMM'`.
`y`	a vector of mode "character" or "raw" (a "DNAbin" or "AAbin" object) representing a single sequence hypothetically emitted by the model in `x`.
`...`	additional arguments to be passed between methods.
`logspace`	logical indicating whether the emission and transition probabilities of `x` are logged. If `logspace = "autodetect"` (default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The R version is primarily retained for bug-fixing and experimentation.

Details

See Durbin et al (1998) chapter 3.2 for details on the calculation and interpretation of posterior state probabilities. Currently no method is available for profile HMMs, but this may be included in a future version if required.

Value

a vector, matrix or array of posterior probabilities.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

See Also

`forward`, `backward`, `Viterbi`

Examples

```
## Posterior decoding for standard hidden Markov models
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino HMM")
### Calculate posterior probabilities
data(casino)
casino.post <- posterior(x, casino)
plot(1:300, casino.post[1, ], type = "l", xlab = "Roll number",
     ylab = "Posterior probability of dice being fair",
     main = "The dishonest casino")
```

`print` *Print summary methods.*

Description

Print summary methods.

Usage

```
## S3 method for class 'PHMM'
print(x, ...)

## S3 method for class 'HMM'
print(x, ...)
```

```
## S3 method for class 'DPA'
print(x, ...)
```

Arguments

`x`	object of various classes.
`...`	additional arguments to be passed between methods.

Value

NULL (invisibly)

Author(s)

Shaun Wilkinson

---

readPHMM — *Import profile hidden Markov models into R.*

Description

The `readPHMM` function parses a HMMER3 text file into R and creates an object of class `"PHMM"`.

Usage

```
readPHMM(file = "", ...)
```

Arguments

`file`	the name of the file from which to read the model.
`...`	further arguments to be passed to `"scan"`.

Details

This function scans a HMMER3/f text file and creates an object of class `"PHMM"` in R. Note that unlike HMMER, the **aphid** package does not currently support position-specific background emission probabilities, and so only a single vector the same length as the reside alphabet is included as an element of the returned object. Also the function currently only parses the first profile HMM encountered in the text file, with subsequent models ignored.

Value

an object of class `"PHMM"`.

Author(s)

Shaun Wilkinson

References

Finn RD, Clements J & Eddy SR (2011) HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*. **39**, W29-W37. http://hmmer.org/.

HMMER: biosequence analysis using profile hidden Markov models. http://www.hmmer.org.

See Also

writePHMM for writing PHMM objects in HMMER3 text format.

Examples

```
## Derive a profile hidden Markov model from the small globin alignment
data(globins)
x <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
fl <- tempfile()
writePHMM(x, file = fl)
readPHMM(fl)
```

`substitution` *Substitution matrices.*

Description

A dataset containing several popular substitution scoring matrices for DNA and amino acids.

Usage

```
substitution
```

Format

A list of 71 matrices, most of which have 24 rows and 24 columns corresponding to the 20-letter amino acid alphabet plus the ambiguity codes B, Z, X and *:

PAM the PAM matrices from PAM10 to PAM500.

BLOSUM the BLOSUM matrices from BLOSUM30 to BLOSUM100.

others also included are the DAYHOFF, GONNET, IDENTITY and MATCH substitution matrices for amino acids, and the NUC.4.2 and NUC.4.4 substitution matrices for DNA.

Source

ftp://ftp.ncbi.nih.gov/blast/matrices/

train *Iterative model refinement.*

Description

Update model parameters using a list of training sequences, with either the Viterbi training or Baum-Welch algorithm.

Usage

```
train(x, y, ...)

## S3 method for class 'PHMM'
train(
  x,
  y,
  method = "Viterbi",
  seqweights = "Henikoff",
  wfactor = 1,
  k = 5,
  logspace = "autodetect",
  maxiter = 100,
  limit = 1,
  deltaLL = 1e-07,
  pseudocounts = "background",
  gap = "-",
  fixqa = FALSE,
  fixqe = FALSE,
  maxsize = NULL,
  inserts = "map",
  threshold = 0.5,
  lambda = 0,
  alignment = FALSE,
  cores = 1,
  quiet = FALSE,
  ...
)

## S3 method for class 'HMM'
train(
  x,
  y,
  method = "Viterbi",
  seqweights = NULL,
  wfactor = 1,
  maxiter = 100,
  deltaLL = 1e-07,
  logspace = "autodetect",
  quiet = FALSE,
  modelend = FALSE,
  pseudocounts = "Laplace",
  ...
)
```

Arguments

`x`	an object of class `"HMM"` or `"PHMM"` specifying the initial parameter values.
`y`	a list of training sequences whose hidden states are unknown. Accepted modes are "character" and "raw" (for "DNAbin" and "AAbin" objects).
`...`	aditional arguments to be passed to `"Viterbi"` (if `method = "Viterbi"`) or `"forward"` (if `method = "BaumWelch"`).
`method`	a character string specifying the iterative model training method to use. Accepted methods are `"Viterbi"` (the default) and `"BaumWelch"`.
`seqweights`	either NULL (all sequences are given weights of 1), a numeric vector the same length as `y` representing the sequence weights used to derive the model, or a character string giving the method to derive the weights from the sequences (see `weight`).
`wfactor`	numeric. The factor to multiply the sequence weights by. Defaults to 1.
`k`	integer representing the k-mer size to be used in tree-based sequence weighting (if applicable). Defaults to 5. Note that higher values of k may be slow to compute and use excessive memory due to the large numbers of calculations required.
`logspace`	logical indicating whether the emission and transition probabilities of `x` are logged. If `logspace = "autodetect"` (default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`maxiter`	the maximum number of EM iterations or Viterbi training iterations to carry out before the cycling process is terminated and the partially trained model is returned. Defaults to 100.
`limit`	the proportion of alignment rows that must be identical between subsequent iterations for the Viterbi training algorithm to terminate. Defaults to 1.
`deltaLL`	numeric, the maximum change in log likelihood between EM iterations before the cycling procedure is terminated (signifying model convergence). Defaults to 1E-07. Only applicable if `method = "BaumWelch"`.
`pseudocounts`	character string, either "background", Laplace" or "none". Used to account for the possible absence of certain transition and/or emission types in the input sequences. If `pseudocounts = "background"` (default), pseudocounts are calculated from the background transition and emission frequencies in the training dataset. If `pseudocounts = "Laplace"` one of each possible transition and emission type is added to the training dataset (default). If `pseudocounts =`

	`"none"` no pseudocounts are added (not usually recommended, since low frequency transition/emission types may be excluded from the model). Alternatively this argument can be a two-element list containing a matrix of transition pseudocounts as its first element and a matrix of emission pseudocounts as its second. If this option is selected, both matrices must have row and column names corresponding with the residues (column names of emission matrix) and states (row and column names of the transition matrix and row names of the emission matrix). For standard HMMs the first row and column of the transition matrix should be named "Begin".
`gap`	the character used to represent gaps in the alignment matrix (if applicable). Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to `"-"` otherwise.
`fixqa`	logical. Should the background transition probabilities be fixed (TRUE), or allowed to vary between iterations (FALSE)? Defaults to FALSE. Only applicable if `method = "Viterbi"`.
`fixqe`	logical. Should the background emission probabilities be fixed (TRUE), or allowed to vary between iterations (FALSE)? Defaults to FALSE. Only applicable if `method = "Viterbi"`.
`maxsize`	integer giving the upper bound on the number of modules in the PHMM. If NULL (default) no maximum size is enforced.
`inserts`	character string giving the model construction method by which alignment columns are marked as either match or insert states. Accepted methods include `"threshold"` (only columns with fewer than a specified proportion of gaps form match states in the model), `"map"` (default; match and insert columns are found using the maximum *a posteriori* method outlined in Durbin et al (1998) chapter 5.7), `"inherited"` (match and insert columns are inherited from the input alignment), and `"none"` (all columns are assigned match states in the model). Alternatively, insert columns can be specified manually by providing a logical vector the same length as the number of columns in the alignment, with `TRUE` for insert columns and `FALSE` for match states.
`threshold`	the maximum proportion of gaps for an alignment column to be considered for a match state in the PHMM (defaults to 0.5). Only applicable when `inserts = "threshold"`. Note that the maximum *a posteriori* method works poorly for alignments with few sequences, so the 'threshold' method is automatically used when the number of sequences is less than 5.
`lambda`	penalty parameter used to favour models with fewer match states. Equivalent to the log of the prior probability of marking each column (Durbin et al 1998, chapter 5.7). Only applicable when `inserts = "map"`.
`alignment`	logical indicating whether the alignment used to derive the final model (if applicable) should be included as an element of the returned PHMM object. Defaults to FALSE.
`cores`	integer giving the number of CPUs to parallelize the operation over. Defaults to 1, and reverts to 1 if x is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string "autodetect" is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in

	this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be aligned, due to the extra time required to initialize the cluster. Only applicable if x is an object of class `"PHMM"`.
`quiet`	logical indicating whether feedback should be printed to the console.
`modelend`	logical indicating whether transition probabilites to the end state of the standard hidden Markov model should be modeled (if applicable). Defaults to FALSE.

Details

This function optimizes the parameters of a hidden Markov model (object class: `"HMM"`) or profile hidden Markov model (object class: `"PHMM"`) using the methods described in Durbin et al (1998) chapters 3.3 and 6.5, respectively. For standard HMMs, the function assumes the state sequence is unknown (as opposed to the `deriveHMM` function, which is used when the state sequence is known). For profile HMMs, the input object is generally a list of non-aligned sequences rather than an alignment (for which the `derivePHMM` function may be more suitable).

This function offers a choice of two model training methods, Viterbi training (also known as the segmental K-means algorithm (Juang & Rabiner 1990)), and the Baum Welch algorithm, a special case of the expectation-maximization (EM) algorithm that iteratively finds the locally (but not necessarily globally) optimal parameters of a HMM or PHMM.

The Viterbi training method is generally much faster, particularly for profile HMMs and when the multi-threading option is used (see the `"cores"` argument). The comparison in accuracy will depend on the nature of the problem, but personal experience suggests that the methods are comparable for training profile HMMs for DNA and amino acid sequences.

Value

an object of class `"HMM"` or `"PHMM"`, depending on the input model `x`.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Juang B-H, Rabiner LR (1990) The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **38**, 1639-1641.

See Also

`deriveHMM` and `derivePHMM` for maximum-likelihood parameter estimation when the training sequence states are known.

Examples

```
## Baum Welch training for standard HMMs:
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
op <- par(no.readonly = TRUE)
par(mfrow = c(2, 1))
plot(x, main = "Dishonest casino HMM before training")
data(casino)
x <- train(x, list(casino), method = "BaumWelch", deltaLL = 0.001)
plot(x, main = "Dishonest casino HMM after training")
par(op)
```

`unalign` *Deconstruct an alignment.*

Description

`unalign` deconstructs an alignment to a list of sequences.

Usage

```
unalign(x, gap = "-")
```

Arguments

`x`	a matrix of aligned sequences. Accepted modes are "character" and "raw" (for "DNAbin" and "AAbin" objects).
`gap`	the character used to represent gaps in the alignment matrix. Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "-" otherwise.

Details

`unalign` works in the opposite way to `align`, reducing a matrix of aligned sequences to a list of sequences without gaps. "DNAbin" and "AAbin" matrix objects are supported (and recommended for biological sequence data)

Value

a list of sequences of the same mode and class as the input alignment (ie "DNAbin", "AAbin", or plain ASCII characters).

Author(s)

Shaun Wilkinson

See Also

`align`.

Examples

```
## Convert the woodmouse alignment in the ape package to a list of
## unaligned sequences
library(ape)
data(woodmouse)
x <- unalign(woodmouse)
```

`Viterbi` *The Viterbi algorithm.*

Description

The `Viterbi` function finds the optimal path of a sequence through a HMM or PHMM and returns its full (log) probability or log-odds score.

Usage

```
Viterbi(x, y, ...)

## S3 method for class 'PHMM'
Viterbi(
  x,
  y,
  qe = NULL,
  logspace = "autodetect",
  type = "global",
  odds = TRUE,
  offset = 0,
  windowspace = "all",
  DI = FALSE,
  ID = FALSE,
  cpp = TRUE,
  ...
)

## S3 method for class 'HMM'
Viterbi(x, y, logspace = "autodetect", cpp = TRUE, ...)

## Default S3 method:
```

```
Viterbi(
  x,
  y,
  type = "global",
  d = 8,
  e = 2,
  residues = NULL,
  S = NULL,
  windowspace = "all",
  offset = 0,
  cpp = TRUE,
  ...
)
```

Arguments

`x`	an object of class `HMM` or `PHMM`. Optionally, both `x` and `y` can be sequences (character vectors or DNAbin/AAbin objects), in which case the operation becomes either the Needleman-Wunch (global algnment) or Smith-Waterman (local alignment) algorithm.
`y`	a vector of mode "character" or "raw" (a "DNAbin" or "AAbin" object) representing a single sequence hypothetically emitted by the model in `x`. Optionally, both `x` and `y` can be profile hidden Markov models (object class "PHMM"), in which case the sum of log-odds algorithm of Soding (2005) is used.
`...`	additional arguments to be passed between methods.
`qe`	an optional named vector of background residue frequencies (only applicable if x is a PHMM). If `qe = NULL` the function looks for a qe vector as an attribute of the PHMM. If these are not available equal background residue frequencies are assumed.
`logspace`	logical indicating whether the emission and transition probabilities of x are logged. If `logspace = "autodetect"` (default setting), the function will automatically detect if the probabilities are logged, returning an error if inconsistencies are found. Note that choosing the latter option increases the computational overhead; therefore specifying `TRUE` or `FALSE` can reduce the running time.
`type`	character string indicating whether insert and delete states at the beginning and end of the path should count towards the final score ('global'; default), or not ('semiglobal'), or whether the highest scoring sub-path should be returned ('local').
`odds`	logical, indicates whether the returned scores should be odds ratios (TRUE) or full logged probabilities (FALSE).
`offset`	column score offset to specify level of greediness. Defaults to -0.1 bits for PHMM x PHMM alignments (as recommended by Soding (2005)), and 0 otherwise.
`windowspace`	a two-element integer vector providing the search space for dynamic programming (see Wilbur & Lipman 1983 for details). The first element should be negative, and represent the lowermost diagonal of the dynammic programming array,

	and the second element should be positive, representing the leftmost diagonal. Alternatively, if the the character string "all" is passed (the default setting) the entire dynamic programming array will be computed.
`DI`	logical indicating whether delete-insert transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`ID`	logical indicating whether insert-delete transitions should be allowed in the profile hidden Markov model (if applicable). Defaults to FALSE.
`cpp`	logical, indicates whether the dynamic programming matrix should be filled using compiled C++ functions (default; many times faster). The FALSE option is primarily retained for bug fixing and experimentation.
`d`	gap opening penalty (in bits) for sequence vs. sequence alignment. Defaults to 8.
`e`	gap extension penalty (in bits) for sequence vs. sequence alignment. Defaults to 2.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors.
`S`	an optional scoring matrix with rows and columns named according to the residue alphabet. Only applicable when both x and y are sequences (Needleman-Wunch or Smith-Waterman alignments). Note that for Smith-Waterman local alignments, scores for mismatches should generally take negative values to avoid spurious alignments. If NULL default settings are used. Default scoring matrices are 'NUC.4.4' for For DNAbin objects, and 'MATCH' (matches are scored 1 and mismatches are scored -1) for AAbin objects and character sequences.

Details

This function is a wrapper for a compiled C++ function that recursively fills a dynamic programming matrix with probabilities, and calculates the (logged) probability and optimal path of a sequence through a HMM or PHMM.

If x is a PHMM and y is a sequence, the path is represented as an integer vector containing zeros, ones and twos, where a zero represents a downward transition, a one represents a diagonal transition downwards and left, and a two represents a left transition in the dynamic programming matrix (see Durbin et al (1998) chapter 2.3). This translates to 0 = delete state, 1 = match state and 2 = insert state.

If x and y are both sequences, the function implements the Needleman-Wunch or Smith Waterman algorithm depending on the type of alignment specified. In this case, a zero in the path refers to x aligning to a gap in y, a one refers to a match, and a two refers to y aligning to a gap in x.

If x is a standard hidden Markov model (HMM) and y is a sequence, each integer in the path represents a state in the model. Note that the path elements can take values between 0 and one less than number of states, as in the C/C++ indexing style rather than R's.

For a thorough explanation of the backward, forward and Viterbi algorithms, see Durbin et al (1998) chapters 3.2 (HMMs) and 5.4 (PHMMs).

Value

an object of class "DPA", which is a list including the score, the dynammic programming array, and the optimal path (an integer vector, see details section).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Soding J (2005) Protein homology detection by HMM-HMM comparison. *Bioinformatics*, **21**, 951-960.

Wilbur WJ, Lipman DJ (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA*, **80**, 726-730.

See Also

backward, forward, align

Examples

```
## Viterbi algorithm for standard HMMs:
## The dishonest casino example from Durbin et al (1998) chapter 3.2
states <- c("Begin", "Fair", "Loaded")
residues <- paste(1:6)
### Define the transition probability matrix
A <- matrix(c(0, 0, 0, 0.99, 0.95, 0.1, 0.01, 0.05, 0.9), nrow = 3)
dimnames(A) <- list(from = states, to = states)
### Define the emission probability matrix
E <- matrix(c(rep(1/6, 6), rep(1/10, 5), 1/2), nrow = 2, byrow = TRUE)
dimnames(E) <- list(states = states[-1], residues = residues)
### Build and plot the HMM object
x <- structure(list(A = A, E = E), class = "HMM")
plot(x, main = "Dishonest casino HMM")
### Find optimal path of sequence
data(casino)
casino.DPA <- Viterbi(x, casino)
casino.DPA$score # full (log) prob of sequence given model = -538.8109
### Show optinal path path as indices
casino.DPA$path
### Show optimal path as character strings
rownames(x$E)[casino.DPA$path + 1]
##
## Needleman-Wunch pairwise sequence alignment:
## Pairwise protein alignment example from Durbin et al (1998) chapter 2.3
x <- c("H", "E", "A", "G", "A", "W", "G", "H", "E", "E")
y <- c("P", "A", "W", "H", "E", "A", "E")
Viterbi(x, y,  d = 8, e = 2, type = "global")
```

```
###
## Viterbi algorithm for profile HMMs:
## Small globin alignment data from Durbin et al (1998) Figure 5.3
data(globins)
### Derive a profile hidden Markov model from the alignment
globins.PHMM <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
plot(globins.PHMM, main = "Profile hidden Markov model for globins")
### Simulate a random sequence from the model
suppressWarnings(RNGversion("3.5.0"))
set.seed(999)
simulation <- generate(globins.PHMM, size = 20)
simulation ## "F" "S" "A" "N" "N" "D" "W" "E"
### Calculate the odds of the optimal path of the sequence given the model
x <- Viterbi(globins.PHMM, simulation, odds = FALSE)
x # -23.07173
### Show dynammic programming array
x$array
### Show the optimal path as an integer vector
x$path
### Show the optimal path as either delete states, matches or insert states
c("D", "M", "I")[x$path + 1]
### Correctly predicted the actual path:
names(simulation)
```

---

weight *Sequence weighting.*

Description

Weighting schemes for DNA and amino acid sequences.

Usage

```
weight(x, ...)

## S3 method for class 'DNAbin'
weight(x, method = "Henikoff", k = 5, ...)

## S3 method for class 'AAbin'
weight(x, method = "Henikoff", k = 5, ...)

## S3 method for class 'list'
weight(x, method = "Henikoff", k = 5, residues = NULL, gap = "-", ...)

## S3 method for class 'dendrogram'
weight(x, method = "Gerstein", ...)

## Default S3 method:
weight(x, method = "Henikoff", k = 5, residues = NULL, gap = "-", ...)
```

Arguments

`x`	a list or matrix of sequences (usually a "DNAbin" or "AAbin" object). Alternatively x can be an object of class `"dendrogram"` for tree-base weighting.
`...`	additional arguments to be passed between methods.
`method`	a character string indicating the weighting method to be used. Currently the only methods available are a modified version of the maximum entropy weighting scheme proposed by Henikoff and Henikoff (1994) (`method = "Henikoff"`) and the tree-based weighting scheme of Gerstein et al (1994) (`method = "Gerstein"`).
`k`	integer representing the k-mer size to be used. Defaults to 5. Note that higher values of k may be slow to compute and use excessive memory due to the large numbers of calculations required.
`residues`	either NULL (default; emitted residues are automatically detected from the sequences), a case sensitive character vector specifying the residue alphabet, or one of the character strings "RNA", "DNA", "AA", "AMINO". Note that the default option can be slow for large lists of character vectors. Furthermore, the default setting `residues = NULL` will not detect rare residues that are not present in the sequences, and thus will not assign them emission probabilities in the model. Specifying the residue alphabet is therefore recommended unless x is a "DNAbin" or "AAbin" object.
`gap`	the character used to represent gaps in the alignment matrix (if applicable). Ignored for `"DNAbin"` or `"AAbin"` objects. Defaults to "-" otherwise.

Details

This is a generic function. If `method = "Henikoff"` the sequences are weighted using a modified version of the maximum entropy method proposed by Henikoff and Henikoff (1994). In this case the maximum entropy weights are calculated from a k-mer presence absence matrix instead of an alignment as originally described by Henikoff and Henikoff (1994). If `method = "Gerstein"` the agglomerative method of Gerstein et al (1994) is used to weight sequences based on their relatedness as derived from a phylogenetic tree. In this case a dendrogram is first derived using the `cluster` function in the `kmer` package. Methods are available for `"dendrogram"` objects, `"DNAbin"` and `"AAbin"` sequence objects (as lists or matrices) and sequences in standard character format provided either as lists or matrices.

For further details on sequence weighting schemes see Durbin et al (1998) chapter 5.8.

Value

a named vector of weights, the sum of which is equal to the total number of sequences (average weight = 1).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Gerstein M, Sonnhammer ELL, Chothia C (1994) Volume changes in protein evolution. *Journal of Molecular Biology*, **236**, 1067-1078.

Henikoff S, Henikoff JG (1994) Position-based sequence weights. *Journal of Molecular Biology*, **243**, 574-578.

Examples

```
## weight the sequences in the woodmouse dataset from the ape package
library(ape)
data(woodmouse)
woodmouse.weights <- weight(woodmouse)
woodmouse.weights
```

---

writePHMM — *Export profile hidden Markov models as text.*

Description

writePHMM takes an object of class "PHMM" and writes it to a text file in HMMER3 format.

Usage

```
writePHMM(x, file = "", append = FALSE, form = "HMMER3", vers = "f")
```

Arguments

x	an object of class "PHMM".
file	the name of the file to write the model to.
append	logical indicating whether the model text should be appended below any existing text in the output file, or whether any existing text should be overwritten. Defaults to FALSE.
form	character string indicating the format in which to write the model. Currently only HMMER3f is supported.
vers	character string indicating the version of version of the format in which to write the model. Currently only "f" is supported.

Details

This function writes an object of class "PHMM" to a HMMER3/f text file. Note that unlike HMMER, the **aphid** package does not currently support position-specific background emission probabilities.

Value

NULL (invisibly)

Author(s)

Shaun Wilkinson

References

Finn, RD, Clements J & Eddy SR (2011) HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*. **39** W29-W37. http://hmmer.org/.

HMMER: biosequence analysis using profile hidden Markov models. http://www.hmmer.org.

See Also

readPHMM to parse a PHMM object from a HMMER3 text file.

Examples

```
## Derive a profile hidden Markov model from the small globin alignment
data(globins)
x <- derivePHMM(globins, residues = "AMINO", seqweights = NULL)
x
fl <- tempfile()
writePHMM(x, file = fl)
readPHMM(fl)
##
## Derive a PHMM for the woodmouse data and write to file

  library(ape)
  data(woodmouse)
  woodmouse.PHMM <- derivePHMM(woodmouse)
  tmpf <- tempfile(fileext = ".hmm")
  writePHMM(woodmouse.PHMM, file = tmpf)
```

Index