Package: DECIPHER (via r-universe)

November 24, 2026

Type Package

Title Tools for curating, analyzing, and manipulating biological sequences

Version 3.10.0

Date 2026-11-18

Author Erik Wright (ORCID: <https://orcid.org/0000-0002-1457-4019>)

Maintainer Erik Wright <Erik.Wright@bcm.edu>

biocViews Clustering, Genetics, Sequencing, DataImport, Visualization, Microarray, QualityControl, qPCR, Alignment, WholeGenome, Microbiome, ImmunoOncology, GenePrediction, Phylogenetics, ComparativeGenomics, SequenceMatching, Classification

Description A toolset for deciphering and managing biological sequences.

Depends R (>= 3.5.0), Biostrings (>= 2.59.1), stats

Imports methods, DBI, S4Vectors, IRanges, XVector

Suggests RSQLite (>= 1.1)

LinkingTo Biostrings, S4Vectors, IRanges, XVector

License GPL-3

ByteCompile true

URL http://DECIPHER.codes

Config/pak/sysreqs zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:31:20 UTC

RemoteUrl https://github.com/bioc/DECIPHER

RemoteRef RELEASE_3_24

RemoteSha a81ec38ec289823432e0e6f81898a63421299bee

Contents

DECIPHER-package . . . 4
AA_REDUCED . . . 7
Add2DB . . . 7
AdjustAlignment . . . 9
AlignDB . . . 11
AlignPairs . . . 14
AlignProfiles . . . 18
AlignSeqs . . . 22
AlignSynteny . . . 25
AlignTranslation . . . 27
AmplifyDNA . . . 29
Array2Matrix . . . 31
BLOSUM . . . 32
BrowseDB . . . 33
BrowseSeqs . . . 35
CalculateEfficiencyArray . . . 39
CalculateEfficiencyFISH . . . 41
CalculateEfficiencyPCR . . . 42
Clusterize . . . 44
Codec . . . 49
ConsensusSequence . . . 51
Cophenetic . . . 53
CorrectFrameshifts . . . 55
CreateChimeras . . . 59
DateTree . . . 60
DB2Seqs . . . 64
deltaGrules . . . 66
deltaGrulesRNA . . . 67
deltaHrules . . . 68
deltaHrulesRNA . . . 69
deltaSrules . . . 70
deltaSrulesRNA . . . 71
DesignArray . . . 72
DesignPrimers . . . 74
DesignProbes . . . 78
DesignSignatures . . . 81
DetectRepeats . . . 85
DigestDNA . . . 89
Disambiguate . . . 90
DistanceMatrix . . . 91
ExtractGenes . . . 98
FindChimeras . . . 99
FindGenes . . . 102
FindNonCoding . . . 104
FindSynteny . . . 105
FormGroups . . . 108

Genes . . . 110
HEC_MI . . . 112
IdConsensus . . . 113
IdentifyByRank . . . 114
IdLengths . . . 116
IdTaxa . . . 117
IndexSeqs . . . 120
InferDemography . . . 123
InferRecombination . . . 125
InferSelection . . . 128
InvertedIndex . . . 131
LearnNonCoding . . . 132
LearnTaxa . . . 134
MapCharacters . . . 138
MaskAlignment . . . 140
MeltDNA . . . 143
MIQS . . . 145
MMLSUM . . . 146
MODELS . . . 147
NNLS . . . 151
NonCoding . . . 152
NonCodingRNA . . . 153
OrientNucleotides . . . 154
PAM . . . 156
PFASUM . . . 157
PredictDBN . . . 158
PredictHEC . . . 162
ReadDendrogram . . . 164
RemoveGaps . . . 165
RESTRICTION_ENZYMES . . . 167
ScoreAlignment . . . 167
SearchDB . . . 170
SearchIndex . . . 172
Seqs2DB . . . 175
StaggerAlignment . . . 177
Synteny . . . 179
Taxa . . . 182
TerminalChar . . . 184
TileSeqs . . . 185
TrainingSet_16S . . . 187
Treeline . . . 188
TrimDNA . . . 196
WriteDendrogram . . . 199
WriteGenes . . . 201
Zipline . . . 202

Index **206**

DECIPHER-package *Tools for curating, analyzing, and manipulating biological sequences*

Description

DECIPHER is a software toolset that can be used for deciphering and managing biological sequences efficiently using the R statistical programming language. The program is designed to be used with non-destructive workflows for importing, maintaining, analyzing, manipulating, and exporting a massive amount of sequences.

Details

Package:	DECIPHER
Type:	Package
Depends:	R (>= 3.5.0), Biostrings (>= 2.59.1), stats, parallel
Imports:	methods, DBI, S4Vectors, IRanges, XVector
Suggests:	RSQLite (>= 1.1)
LinkingTo:	Biostrings, S4Vectors, IRanges, XVector
License:	GPL-3
LazyLoad:	yes

Index:

AA_REDUCED	Reduced amino acid alphabets
Add2DB	Add Data to a Database
AdjustAlignment	Improve An Existing Alignment By Adjusting Gap Placements
AlignDB	Align Two Sets of Aligned Sequences in a Sequence Database
AlignPairs	Align pairs of sequences
AlignProfiles	Align Two Sets of Aligned Sequences
AlignSeqs	Align a Set of Unaligned Sequences
AlignSynteny	Pairwise Aligns Syntenic Blocks
AlignTranslation	Align Sequences By Their Amino Acid Translation
AmplifyDNA	Simulate Amplification of DNA by PCR
Array2Matrix	Create a Matrix Representation of a Microarray
BLOSUM	BLOSUM Amino Acid Substitution Matrices
BrowseDB	View a Database Table in a Web Browser
BrowseSeqs	View Sequences in a Web Browser
CalculateEfficiencyArray	Predict the Hybridization Efficiency of Probe/Target Sequence Pairs
CalculateEfficiencyFISH	Predict Thermodynamic Parameters of Probe/Target Sequence Pairs
CalculateEfficiencyPCR	Predict Amplification Efficiency of Primer Sequences
Clusterize	Cluster Sequences By Distance

Codec	Compression/Decompression of Character Vectors
ConsensusSequence	Create a Consensus Sequence
Cophenetic	Compute Cophenetic Distances on Dendrogram Objects
CorrectFrameshifts	Corrects Frameshift Errors In Protein Coding Sequences
CreateChimeras	Create Artificial Chimeras
DateTree	Time Calibrate a Dendrogram
DB2Seqs	Export Database Sequences to a FASTA or FASTQ File
deltaGrules	Free Energy of Hybridization of Probe/Target Quadruplets
deltaHrules	Change in Enthalpy of Hybridization of DNA/DNA Quadruplets in Solution
deltaHrulesRNA	Change in Enthalpy of Hybridization of RNA/RNA Quadruplets in Solution
deltaSrules	Change in Entropy of Hybridization of DNA/DNA Quadruplets in Solution
deltaSrulesRNA	Change in Entropy of Hybridization of RNA/RNA Quadruplets in Solution
DesignArray	Design a Set of DNA Microarray Probes for Detecting Sequences
DesignPrimers	Design Primers Targeting a Specific Group of Sequences
DesignProbes	Design FISH Probes Targeting a Specific Group of Sequences
DesignSignatures	Design PCR Primers for Amplifying Group-Specific Signatures
DetectRepeats	Detect Repeats in a Sequence
DigestDNA	Simulate Restriction Digestion of DNA
Disambiguate	Expand Ambiguities into All Permutations of a DNAStringSet
DistanceMatrix	Calculate the Distance Between Sequences
ExtractGenes	Extract Predicted Genes from a Genome
FindChimeras	Find Chimeras in a Sequence Database
FindGenes	Find Genes in a Genome
FindNonCoding	Find Non-Coding RNAs in a Genome
FindSynteny	Finds Synteny in a Sequence Database
FormGroups	Forms Groups By Rank
Genes-class	Genes objects and accessors
HEC_MI	Mutual Information for Protein Secondary Structure Prediction
IdConsensus	Create Consensus Sequences by Groups
IdentifyByRank	Identify By Taxonomic Rank
IdLengths	Determine the Number of Characters in Each Sequence of Each Sequence
IdTaxa	Assign Sequences a Taxonomic Classification
IndexSeqs	Build an inverted index
InferDemography	Infer Demographic History from Allele Frequencies
InferRecombination	Infer Recombination Parameters from Correlation Profiles

InferSelection	Infer Codon Selection on Protein Coding Sequences
InvertedIndex-class	InvertedIndex objects
LearnNonCoding	Learn a Non-Coding RNA Model
LearnTaxa	Train a Classifier for Assigning Taxonomy
MapCharacters	Map Changes in Ancestral Character States
MaskAlignment	Mask Highly Variable Regions of An Alignment
MeltDNA	Simulate Melting of DNA
MIQS	MIQS Amino Acid Substitution Matrix
MMLSUM	MMLSUM Amino Acid Substitution Matrices
MODELS	Available Models of Sequence Evolution
NNLS	Sequential Coordinate-wise Algorithm for the Non-negative Least Squares Problem
NonCoding	NonCoding Models for Common Non-Coding RNA Families
NonCoding-class	NonCoding Objects and Methods
OrientNucleotides	Orient Nucleotide Sequences
PAM	PAM Amino Acid Substitution Matrices
PFASUM	PFASUM Amino Acid Substitution Matrices
PredictDBN	Predict RNA Secondary Structure in Dot-Bracket Notation
PredictHEC	Predict Protein Secondary Structure
Read Dendrogram	Read a Dendrogram from a Newick Formatted File
RemoveGaps	Remove Gap Characters in Sequences
RESTRICTION_ENZYMES	Common Restriction Enzyme's Cut Sites
ScoreAlignment	Score a Multiple Sequence Alignment
SearchDB	Obtain Specific Sequences from a Database
SearchIndex	Search an inverted index
Seqs2DB	Add Sequences from Text File to Database
StaggerAlignment	Produce a Staggered Alignment
Synteny-class	Synteny blocks and hits
Taxa-class	Taxa training and testing objects
TerminalChar	Determine the Number of Terminal Characters
TileSeqs	Form a Set of Tiles for Each Group of Sequences
TrainingSet_16S	Training Set for Classification of 16S rRNA Gene Sequences
Treeline	Construct a Phylogenetic Tree
TrimDNA	Trims DNA Sequences to the High Quality Region Between Patterns
WriteDendrogram	Write a Dendrogram to Newick Format
WriteGenes	Write Genes to a File
Zipline	Summarize Multiple Phylogenetic Trees

Author(s)

Erik Wright

Maintainer: Erik Wright <Erik.Wright@bcm.edu>

AA_REDUCED *Reduced amino acid alphabets*

Description

The AA_REDUCED list contains reductions of the standard amino acid alphabet (AA_STANDARD).

Usage

```
AA_REDUCED
```

Details

Reduced amino acid alphabets can sometimes improve sensitivity and specificity of finding homologous matches between amino acid sequences. The first 12 AA_REDUCED alphabets were optimized for finding synteny between genomic sequences. The next 113 alphabets are from a review of published amino acid alphabets (Solis, 2015). The following 17 alphabets were optimized for amino acid classification. The subsequent 18 alphabets are progressive mergers based on average similarities in PFASUM. The following 25 alphabets were optimized for protein search. The final 8 alphabets were optimized for clustering.

References

Solis, A. (2015). Amino acid alphabet reduction preserves fold information contained in contact interactions in proteins. *Proteins: Structure, Function, and Genetics*, **83(12)**, 2198-2216.

See Also

FindSynteny, LearnTaxa, PFASUM

Examples

```
str(AA_REDUCED)
AA_REDUCED[[1]]
```

Add2DB *Add Data to a Database*

Description

Adds a data.frame to a database table by its row.names.

Usage

```
Add2DB(myData,
       dbFile,
       tblName = "Seqs",
       clause = "",
       verbose = TRUE)
```

Arguments

myData	Data frame containing information to be added to the dbFile.
dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table in which to add the data.
clause	An optional character string to append to the query as part of a "where clause".
verbose	Logical indicating whether to display each query as it is sent to the database.

Details

Data contained in myData will be added to the tblName by its respective row.names.

Value

Returns TRUE if the data was added successfully, or FALSE otherwise.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

See Also

Seqs2DB, SearchDB, BrowseDB

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 # Create a sequence database
 gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
 dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
 Seqs2DB(gen, "GenBank", dbConn, "Bacteria")

 # Identify the sequence lengths
 l <- IdLengths(dbConn)

 # Add lengths to the database
```

```
Add2DB(l, dbConn)

# View the added lengths
BrowseDB(dbConn)

# Change the value of existing columns
ids <- data.frame(identifier=rep("Bacteroidetes", 18), stringsAsFactors=FALSE)
rownames(ids) <- 10:27
Add2DB(ids, dbConn)
BrowseDB(dbConn)

# Add data to a subset of rows using a clause
ids[[1]][] <- "Changed"
nrow(ids) # 18 rows
Add2DB(ids, dbConn, clause="accession like 'EU808318%'")
BrowseDB(dbConn) # only 1 row effected

dbDisconnect(dbConn)
}
```

`AdjustAlignment` *Improve An Existing Alignment By Adjusting Gap Placements*

Description

Makes small adjustments by shifting groups of gaps left and right to find their optimal positioning in a multiple sequence alignment.

Usage

```
AdjustAlignment(myXStringSet,
                perfectMatch = 2,
                misMatch = -1,
                gapLetter = -3,
                gapOpening = -0.1,
                gapExtension = 0,
                substitutionMatrix = NULL,
                shiftPenalty = -0.2,
                threshold = 0.1,
                weight = 1,
                processors = 1)
```

Arguments

`myXStringSet`	An `AAStringSet`, `DNAStringSet`, or `RNAStringSet` object of aligned sequences.
`perfectMatch`	Numeric giving the reward for aligning two matching nucleotides in the alignment. Only used for `DNAStringSet` or `RNAStringSet` inputs.

misMatch — Numeric giving the cost for aligning two mismatched nucleotides in the alignment. Only used for DNAStringSet or RNAStringSet inputs.

gapLetter — Numeric giving the cost for aligning gaps to letters. A lower value (more negative) encourages the overlapping of gaps across different sequences in the alignment.

gapOpening — Numeric giving the cost for opening or closing a gap in the alignment.

gapExtension — Numeric giving the cost for extending an open gap in the alignment.

substitutionMatrix — Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment. The default (NULL) will use the perfectMatch and misMatch penalties for DNA/RNA or PFASUM50 for AA.

shiftPenalty — Numeric giving the cost for every additional position that a group of gaps is shifted.

threshold — Numeric specifying the improvement in score required to permanently apply an adjustment to the alignment.

weight — A numeric vector of weights for each sequence, or a single number implying equal weights.

processors — The number of processors to use, or NULL to automatically detect and use all available processors.

Details

The process of progressive multiple sequence alignment often results in the integration of imperfections into the final alignment. Some of these errors are obvious by eye, which encourages manual refinement of automatically generated alignments. However, the manual refinement process is inherently subjective and time consuming. AdjustAlignment refines an existing alignment in a process similar to that which might be applied manually, but in a repeatable and faster fashion. This function iteratively shifts all of the gaps in an alignment to the left and right to find their optimal positioning. The optimal position is defined as the position that maximizes the alignment "score", which is determined by the input parameters. The resulting alignment will be similar to the input alignment but with many imperfections eliminated. Note that the affine gap penalties here are different from the more flexible penalties used in AlignProfiles, and have been optimized independently.

Value

An XStringSet of aligned sequences.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics, 16, 322. http://doi.org/10.1186/s12859-015-0749-z

Wright, E. S. (2020). RNAconTest: comparing tools for noncoding RNA multiple sequence alignment based on structural consistency. RNA 2020, 26, 531-540.

See Also

AlignSeqs, AlignTranslation, PFASUM, StaggerAlignment

Run `vignette("ArtOfAlignmentInR", package = "DECIPHER")` to see a related vignette.

Examples

```
# a trivial example
aa <- AAStringSet(c("ARN-PK", "ARRP-K"))
aa # input alignment
AdjustAlignment(aa) # output alignment

# specifying an alternative substitution matrix
AdjustAlignment(aa, substitutionMatrix="BLOSUM62")

# a real example
fas <- system.file("extdata", "Streptomyces_ITS_aligned.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna # input alignment
adjustedDNA <- AdjustAlignment(dna) # output alignment
BrowseSeqs(adjustedDNA, highlight=1)
adjustedDNA==dna # most sequences were adjusted (those marked FALSE)
```

AlignDB *Align Two Sets of Aligned Sequences in a Sequence Database*

Description

Merges the two separate sequence alignments in a database. The aligned sequences must have separate identifiers in the same table or be located in different database tables.

Usage

```
AlignDB(dbFile,
        tblName = "Seqs",
        identifier = "",
        type = "DNAStringSet",
        add2tbl = "Seqs",
        batchSize = 10000,
        perfectMatch = 2,
        misMatch = -1,
        gapOpening = -12,
        gapExtension = -3,
        gapPower = -1,
        terminalGap = -4,
```

```
normPower = c(1, 0),
standardize = TRUE,
substitutionMatrix = NULL,
processors = 1,
verbose = TRUE,
...)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table(s) where the sequences are located. If two tblNames are provided then the sequences in both tables will be aligned.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected. If two identifiers are provided then the set of sequences matching each identifier will be aligned.
type	The type of XStringSet being processed. This should be one of "AAStringSet", "DNAStringSet", or "RNAStringSet".
add2tbl	Character string specifying the table name in which to add the aligned sequences.
batchSize	Integer specifying the number of sequences to process at a time.
perfectMatch	Numeric giving the reward for aligning two matching nucleotides in the alignment. Only used when type is DNAStringSet or RNAStringSet.
misMatch	Numeric giving the cost for aligning two mismatched nucleotides in the alignment. Only used when type is DNAStringSet or RNAStringSet.
gapOpening	Numeric giving the cost for opening a gap in the alignment.
gapExtension	Numeric giving the cost for extending an open gap in the alignment.
gapPower	Numeric specifying the exponent to use in the gap cost function.
terminalGap	Numeric giving the cost for allowing leading and trailing gaps ("-" or "." characters) in the alignment. Either two numbers, the first for leading gaps and the second for trailing gaps, or a single number for both.
normPower	Numeric giving the exponent that controls the degree of normalization applied to scores by column occupancy. If two numerics are provided, the first controls the normalization power of terminal gaps, while the second controls that of internal gaps. A normPower of 0 does not normalize the scores, which results in all columns of the profiles being weighted equally, and is the optimal value for aligning fragmentary sequences. A normPower of 1 normalizes the score for aligning two positions by their column occupancy (1 - fraction of gaps). A normPower greater than 1 more strongly discourages aligning with "gappy" regions of the alignment.
standardize	Logical determining whether scores are standardized to be in units of per matching site. Standardization effectively divides the score of each possible alignment by its length so that scores are relative rather than absolute.

substitutionMatrix
: Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment. The default (NULL) will use the `perfectMatch` and `misMatch` penalties for DNA/RNA or `PFASUM50` for AA.

processors
: The number of processors to use, or `NULL` to automatically detect and use all available processors.

verbose
: Logical indicating whether to display progress.

...
: Further arguments to be passed directly to `Codec`.

Details

Sometimes it is useful to align two large sets of sequences, where each set of sequences is already aligned but the two sets are not aligned to each other. `AlignDB` first builds a profile of each sequence set in increments of `batchSize` so that the entire sequence set is not required to fit in memory. Next the two profiles are aligned using dynamic programming. Finally, the new alignment is applied to all the sequences as they are incrementally added to the `add2tbl`.

Two `identifiers` or `tblNames` must be provided, indicating the two sets of sequences to align. The sequences corresponding to the first `identifier` and `tblName` will be aligned to those of the second `identifier` or `tblName`. The aligned sequences are added to `add2tbl` under a new identifier formed from the concatenation of the two `identifiers` or `tblNames`. (See examples section below.)

Value

Returns the number of newly aligned sequences added to the database.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics, 16, 322. http://doi.org/10.1186/s12859-015-0749-z

Wright, E. S. (2020). RNAconTest: comparing tools for noncoding RNA multiple sequence alignment based on structural consistency. RNA 2020, 26, 531-540.

See Also

`AlignProfiles`, `AlignSeqs`, `AlignTranslation`, `PFASUM`

Run `vignette("ArtOfAlignmentInR", package = "DECIPHER")` to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
 fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
```

```
# Align two tables and place result into a third
dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
Seqs2DB(gen, "GenBank", dbConn, "Seqs1", tblName="Set1")
Seqs2DB(fas, "FASTA", dbConn, "Seqs2", tblName="Set2")
AlignDB(dbConn, tblName=c("Set1", "Set2"), add2tbl="AlignedSets")
l <- IdLengths(dbConn, "AlignedSets", add2tbl=TRUE)
BrowseDB(dbConn, tblName="AlignedSets") # all sequences have the same width
dbDisconnect(dbConn)

# Align two identifiers and place the result in the same table
dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
Seqs2DB(gen, "GenBank", dbConn, "Seqs1")
Seqs2DB(fas, "FASTA", dbConn, "Seqs2")
AlignDB(dbConn, identifier=c("Seqs1", "Seqs2"))
l <- IdLengths(dbConn, add2tbl=TRUE)
BrowseDB(dbConn) # note the sequences with a new identifier
dbDisconnect(dbConn)
}
```

`AlignPairs` *Align pairs of sequences*

Description

Aligns pairs of sequences globally or locally using anchored adaptive banding.

Usage

```
AlignPairs(pattern,
           subject,
           pairs = NULL,
           type = "values",
           perfectMatch = 2,
           misMatch = -1,
           gapOpening = -16,
           gapExtension = -1.2,
           substitutionMatrix = NULL,
           bandWidth = 50,
           dropScore = -100,
           processors = 1,
           verbose = TRUE)
```

Arguments

`pattern`	An `AAStringSet`, `DNAStringSet`, or `RNAStringSet` object of (unaligned) sequences, or a `QualityScaledXStringSet` object with quality scores for each sequence.

subject — A XStringSet or QualityScaledXStringSet object of (unaligned) sequences matching the type of the pattern.

pairs — Either NULL or a data.frame with Pattern and Subject indices to align and, optionally, a Position column containing matrices of anchor ranges. If Position is provided then local alignment is performed around the anchors, unless virtual anchors are provided to force global alignment with terminal gap penalties. If pairs is NULL (the default) then global alignment of respective pattern and subject sequences is performed without terminal gap penalties. (See examples section below.)

type — Character string indicating the type of output desired. This should be one of "values", "sequences", or "both". (See value section below.)

perfectMatch — Numeric giving the reward for aligning matching nucleotides, which is used in the absence of a substitutionMatrix when the pattern is a DNAStringSet or RNAStringSet.

misMatch — Numeric determining the cost for aligning mismatched nucleotides, which is used in the absence of a substitutionMatrix when the pattern is a DNAStringSet or RNAStringSet.

gapOpening — Numeric giving the cost for opening a gap in the alignment.

gapExtension — Numeric giving the cost for extending an open gap in the alignment.

substitutionMatrix — Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment. The default (NULL) will use the perfectMatch and misMatch penalties for DNA/RNA or PFASUM50 for amino acids including "U" (selenocysteine) and "O" (pyrrolysine) with scores of zero.

bandWidth — Integer determining the number of positions included in the adaptive band, which should be at least as large as the largest expected insertion or deletion (i.e., gap). Smaller values will accelerate alignment, potentially at the expense of accuracy.

dropScore — Numeric giving the decrease in score required to stop extending the region to the left or right of flanking anchors when performing local alignment. Lower values find longer alignments at the expense of speed.

processors — The number of processors to use, or NULL to automatically detect and use all available processors.

verbose — Logical indicating whether to display progress.

Details

Performs pairwise alignment of pattern and subject sequences, either in their respective pairs or as specified by pairs. Uses adaptive banding and (optionally) anchoring to accelerate the alignment. Unlike AlignProfiles, AlignPairs can perform local alignment around anchor positions to align local regions of the pattern and/or subject sequences. In the absence of anchors, AlignPairs performs global alignment without terminal gap penalties or with terminal gap penalties when virtual anchors are provided immediately outside the bounds of the sequences. (See examples section below.)

AlignPairs is designed to maximize speed, and provides slightly less accuracy than using AlignProfiles for pairwise alignment. Adaptive banding is applied with a fixed bandWidth to reduce memory

consumption, rather than the dynamic band width used by AlignProfiles. There are a few other differences: AlignPairs applies affine rather than Zipfian gap penalties, there is no option to incorporate secondary structures, scores are not standardized by length, gap penalties are not modulated around specific residues, and any anchors must be supplied in pairs rather than determined automatically. For very dissimilar sequences, it is preferable to use AlignTranslation (best for coding sequences), AlignSeqs (best for nucleotide/protein sequences), or AlignProfiles.

AlignPairs does not output the pairwise alignments by default. Instead, it outputs statistics about the alignment and the position(s) of gaps in each sequence. This makes alignment more efficient because no sequences are copied. For many applications only the percent identity or number of gaps is needed, which can be calculated directly from the returned data.frame. However, the aligned sequences can also easily be obtained by setting type to "sequences". Doing so will also output a consensus sequence for each pairwise alignment, which takes into account quality scores when pattern and subject are QualityScaledXStringSets. Note that letters take precedence over gaps in the consensus sequence, resulting in a gapless consensus unless the provided quality scores imply an internal gap is higher quality. This allows the merger of overlapping (e.g., paired-end) sequence reads based on their associated quality scores.

The output consensus sequences will have associated quality scores when the input sequences are QualityScaledXStringSets. The quality scores represent the posterior probability of error based on whether the aligned positions agree and their corresponding qualities. For example, if the aligned positions agree and have high quality scores, then the consensus position will have an even higher quality score. If two aligned positions disagree and have high quality scores, then the consensus will have a low quality score. The output provides a representation of error along the consensus sequences so long as the input quality scores accurately represent the probability of error.

Value

If type is "values" (the default), a data.frame is returned with one row of results per input pattern or row of pairs if not NULL. Columns are defined as the sequences' index in pattern (Pattern), start position in the pattern sequence (PatternStart), end position in the pattern sequence (PatternEnd), index in the subject sequence (Subject), start position in the subject sequence (SubjectStart), end position in the subject sequence (SubjectEnd), number of matching positions in the alignment (Matches), number of mismatched positions in the alignment (Mismatches), total number of positions in the alignment (AlignmentLength), alignment score (Score), and the position/length of gaps in the pattern and subject (i.e., PatternGapPosition, PatternGapLength, SubjectGapPosition, & SubjectGapLength).

If type is "sequences", a list containing three components: the aligned pattern, subject, and their consensus.

If type is "both", a list with four components: the result values, aligned pattern, aligned subject, and their consensus.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2024) "Fast and Flexible Search for Homologous Biological Sequences with DECIPHER v3". The R Journal, **16(2)**, 191-200.

See Also

AlignProfiles, ConsensusSequence, IndexSeqs, SearchIndex

Run vignette("SearchForResearch", package = "DECIPHER") to see a related vignette.

Examples

```
# import target sequences and build an inverted index
fas <- system.file("extdata", "PlanctobacteriaNamedGenes.fas.gz", package="DECIPHER")
target <- readAAStringSet(fas)
index <- IndexSeqs(target, K=6L)
index

# import query sequences and search the index
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
query <- translate(dna)
hits <- SearchIndex(query, index, scoreOnly=FALSE)
head(hits)
nrow(hits) # number of query/target pairs

# locally pairwise align the query/target pairs representing each hit
aligned <- AlignPairs(query, target, hits) # local alignment around hits
head(aligned)

# plot the percent identity of each alignment versus score
PIDs <- aligned$Matches/aligned$AlignmentLength
plot(PIDs, hits$Score) # versus the hit score
plot(PIDs, aligned$Score) # versus the alignment score

# plot the number of pattern gaps versus subject gaps per alignment
subjectGaps <- sapply(aligned$SubjectGapLength, sum)
patternGaps <- sapply(aligned$PatternGapLength, sum)
plot(subjectGaps, patternGaps, pch=16, col="#00000011")
abline(a=0, b=1) # identity line (y = x)

# extract the pairwise aligned regions
alignments <- AlignPairs(query, target, hits, type="sequences")
BrowseSeqs(c(alignments$Pattern[1], alignments$Subject[1])) # view the first pair

# perform global pairwise alignment by creating virtual endpoint anchors
# virtual anchors are immediately out-of-bounds (positions 0 and width + 1)
# this causes gap opening/extension penalties to be applied at each terminus
hits$Position <- mapply(function(x, y, z)
  cbind(matrix(0L, 4), x, matrix(c(y, y, z, z), 4)),
 hits$Position,
 width(query)[hits$Pattern] + 1L,
 width(target)[hits$Subject] + 1L,
 SIMPLIFY=FALSE)
aligned <- AlignPairs(query, target, hits) # penalizes terminal gaps
head(aligned) # all alignments now span start to end of the sequences

# perform global pairwise alignment of sequences without anchors (approach 1)
```

```
pattern <- query[rep(1, length(query))] # first sequence repeated
subject <- query
aligned1 <- AlignPairs(pattern, subject) # no terminal gap penalties
head(aligned1)
# perform global pairwise alignment of sequences without anchors (approach 2)
pairs <- data.frame(Pattern=1L, Subject=seq_along(query))
aligned2 <- AlignPairs(query, query, pairs) # no terminal gap penalties
head(aligned2) # note the Pattern column is always 1 (first sequence)

# merge paired-end reads
faq1 <- system.file("extdata", "Simulated_Illumina_Read1.fq.gz", package="DECIPHER")
faq2 <- system.file("extdata", "Simulated_Illumina_Read2.fq.gz", package="DECIPHER")
read1 <- readQualityScaledDNAStringSet(faq1)
read2 <- readQualityScaledDNAStringSet(faq2)
merge <- AlignPairs(read1,
 reverseComplement(read2),
 type="sequences")
merge$Consensus
```

AlignProfiles *Align Two Sets of Aligned Sequences*

Description

Aligns two sets aligned sequences by first generating representative profiles, then aligning the profiles with dynamic programming, and finally merging the two aligned sequence sets.

Usage

```
AlignProfiles(pattern,
              subject,
              p.weight = 1,
              s.weight = 1,
              p.struct = NULL,
              s.struct = NULL,
              perfectMatch = 2,
              misMatch = -1,
              gapOpening = -12,
              gapExtension = -3,
              gapPower = -1,
              terminalGap = -4,
              restrict = c(-1000, 2, 10),
              anchor = 0.7,
              normPower = c(1, 0),
              standardize = TRUE,
              substitutionMatrix = NULL,
              structureMatrix = NULL,
              processors = 1)
```

Arguments

pattern	An AAStringSet, DNAStringSet, or RNAStringSet object of aligned sequences to use as the pattern.
subject	A XStringSet object of aligned sequences to use as the subject. Must match the type of the pattern.
p.weight	A numeric vector of weights for each sequence in the pattern to use in generating a profile, or a single number implying equal weights.
s.weight	A numeric vector of weights for each sequence in the subject to use in generating a profile, or a single number implying equal weights.
p.struct	Either NULL (the default), a matrix, or a list of matrices with one list element per sequence in the pattern. (See details section below.)
s.struct	Either NULL (the default), a matrix, or a list of matrices with one list element per sequence in the subject. (See details section below.)
perfectMatch	Numeric giving the reward for aligning two matching nucleotides in the alignment. Only applicable for DNAStringSet or RNAStringSet inputs.
misMatch	Numeric giving the cost for aligning two mismatched nucleotides in the alignment. Only applicable for DNAStringSet or RNAStringSet inputs.
gapOpening	Numeric giving the cost for opening a gap in the alignment.
gapExtension	Numeric giving the cost for extending an open gap in the alignment.
gapPower	Numeric specifying the exponent to use in the gap cost function. (See details section below.)
terminalGap	Numeric giving the cost for allowing leading and trailing gaps ("-" or "." characters) in the alignment. Either two numbers, the first for leading gaps and the second for trailing gaps, or a single number for both.
restrict	Numeric vector of length three controlling the degree of restriction around ridge lines in the dynamic programming matrix. The first element determines the span of the region around a ridge that is considered during alignment. The default (-1000) will align most inputs that can reasonably be globally aligned without any loss in accuracy. Input sequences with high similarity could be more restricted (e.g., -500), whereas a pattern and subject with little overlap should be less restricted (e.g., -10000). The second element sets the minimum slope to either side of a ridge that is required to allow restriction at any point. The third element sets the minimum duration of the ridge required to begin restricting the matrix around the ridge. The duration of the ridge is defined as the number of consecutive positions meeting the first two conditions for restriction. (See details section below.)
anchor	Numeric giving the fraction of sequences with identical k-mers required to become an anchor point, or NA to not use anchors. Alternatively, a matrix specifying anchor regions. (See details section below.)
normPower	Numeric giving the exponent that controls the degree of normalization applied to scores by column occupancy. If two numerics are provided, the first controls the normalization power of terminal gaps, while the second controls that of internal gaps. A normPower of 0 does not normalize the scores, which results in all columns of the profiles being weighted equally, and is the optimal value

for aligning fragmentary sequences. A normPower of 1 normalizes the score for aligning two positions by their column occupancy (1 - fraction of gaps). A normPower greater than 1 more strongly discourages aligning with "gappy" regions of the alignment. (See details section below.)

standardize — Logical determining whether scores are standardized to be in units of per matching site. Standardization effectively divides the score of each possible alignment by its length so that scores are relative rather than absolute.

substitutionMatrix — Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment. The default (NULL) will use the perfectMatch and misMatch penalties for DNA/RNA or PFASUM50 for AA. (See examples section below.)

structureMatrix — A structure matrix if p.struct and s.struct are supplied, or NULL otherwise.

processors — The number of processors to use, or NULL to automatically detect and use all available processors.

Details

Profiles are aligned using dynamic programming, a variation of the Needleman-Wunsch algorithm for global alignment. The dynamic programming method requires order N*M time and memory space where N and M are the width of the pattern and subject. This method works by filling in a matrix of the possible "alignment space" by considering all matches, insertions, and deletions between two sequence profiles. The highest scoring alignment is then used to add gaps to each of the input sequence sets.

Heuristics can be useful to improve performance on long input sequences. The restrict parameter can be used to dynamically constrain the possible "alignment space" to only paths that will likely include the final alignment, which in the best case can improve the speed from quadratic time to nearly linear time. The degree of restriction is important, and the default value of restrict is reasonable in the vast majority of cases. It is also possible to prevent restriction by setting restrict to such extreme values that these requirements will never be met (e.g., c(-1e10, 1e10, 1e10)).

The argument anchor can be used to split the global alignment into multiple sub-alignments. This can greatly decrease the memory requirement for long sequences when appropriate anchor points can be found. Anchors are 15-mer (for DNA/RNA) or 7-mer (for AA) subsequences that are shared between at least anchor fraction of pattern(s) and subject(s). Anchored ranges are extended along the length of each sequence in a manner designed to split the alignment into sub-alignments that can be separately solved. For most input sequences, the default anchoring has no effect on accuracy, but anchoring can be disabled by setting anchor=NA.

Alternatively, anchor can be a matrix with 4 rows and one column per anchor. The first two rows correspond to the anchor start and end positions in the pattern sequence(s), and the second two rows are the equivalent anchor region in the subject sequence(s). Anchors specified in this manner must be strictly increasing (non-overlapping) in both sequences, and have an anchor width of at least two positions. Note that the anchors do not have to be equal length, in which case the anchor regions will also be aligned. Manually splitting the alignment into more subtasks can sometimes make it more efficient, but typically automatic anchoring is effective.

The argument normPower determines how the distribution of information is treated during alignment. Higher values of normPower encourage alignment between columns with higher occupancy

(1 - fraction of gaps), and de-emphasize the alignment of columns containing many gaps. A `normPower` of `0` will treat all columns equally regardless of occupancy, which can be useful when the `pattern` or `subject` contain many incomplete (fragment) sequences. For example, `normPower` should be set to `0` when aligning many short reads to a longer reference sequence.

The arguments `p.struct` and `s.struct` may be used to provide secondary structure probabilities in the form of a list containing matrices or a single matrix. If the input is a list, then each list element must contain a matrix with dimensions `q*w`, where `q` is the number of possible secondary structure states, and `w` is the width of the unaligned pattern sequence. Values in each matrix represent the probability of the given state at that position in the sequence. Alternatively, a single matrix can be used as input if `w` is the width of the entire `pattern` or `subject` alignment. A `structureMatrix` must be supplied along with the structures. The functions `PredictHEC` and `PredictDBN` can be used to predict secondary structure probabilities in the format required by `AlignProfiles` (for amino acid and RNA sequences, respectively).

The gap cost function is based on the observation that gap lengths are best approximated by a Zipfian distribution (Chang & Benner, 2004). The cost of inserting a gap of length `L` is equal to: `gapOpening + gapExtension*sum(seq_len(L - 1)^gapPower)` when `L > 1`, and `gapOpen` when `L = 1`. This function effectively penalizes shorter gaps significantly more than longer gaps when `gapPower < 0`, and is equivalent to the affine gap penalty when `gapPower` is `0`.

Value

An `XStringSet` of aligned sequences.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Chang, M. S. S., & Benner, S. A. (2004). Empirical Analysis of Protein Insertions and Deletions Determining Parameters for the Correct Placement of Gaps in Protein Sequence Alignments. Journal of Molecular Biology, **341(2)**, 617-631.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, **48(3)**, 443-453.

Wright, E. S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics, 16, 322. http://doi.org/10.1186/s12859-015-0749-z

Wright, E. S. (2020). RNAconTest: comparing tools for noncoding RNA multiple sequence alignment based on structural consistency. RNA 2020, 26, 531-540.

Yu, Y.-K., et al. (2015). Log-odds sequence logos. Bioinformatics, 31(3), 324-331. http://doi.org/10.1093/bioinformatics/btu

See Also

`AlignDB`, `AlignSeqs`, `AlignSynteny`, `AlignTranslation`, `PFASUM`, `MIQS`

Run `vignette("ArtOfAlignmentInR", package = "DECIPHER")` to see a related vignette.

Examples

```
# align two sets of sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna1 <- readDNAStringSet(fas, n=100) # the first 100 sequences
dna2 <- readDNAStringSet(fas, n=100, skip=100) # the rest
dna1 <- RemoveGaps(dna1, "common")
dna2 <- RemoveGaps(dna2, "common")
alignedDNA <- AlignProfiles(dna1, dna2)
BrowseSeqs(alignedDNA, highlight=1)

# specify a DNA substitution matrix
subMatrix <- matrix(0,
                    nrow=4, ncol=4,
                    dimnames=list(DNA_BASES, DNA_BASES))
diag(subMatrix) <- 5 # perfectMatch
alignedDNA.defaultSubM <- AlignProfiles(dna1, dna2, substitutionMatrix=subMatrix)
all(alignedDNA.defaultSubM==alignedDNA)

# specify a different DNA substitution matrix
subMatrix2 <- matrix(c(12, 3, 5, 3, 3, 12, 3, 6, 5, 3, 11, 3, 3, 6, 3, 9),
                    nrow=4, ncol=4,
                    dimnames=list(DNA_BASES, DNA_BASES))
alignedDNA.alterSubM <- AlignProfiles(dna1, dna2, substitutionMatrix=subMatrix2)
all(alignedDNA.alterSubM==alignedDNA)

# anchors are found automatically by default, but it is also
# possible to specify anchor regions between the sequences
anchors <- matrix(c(774, 788, 752, 766), nrow=4)
anchors
subseq(dna1, anchors[1, 1], anchors[2, 1])
subseq(dna2, anchors[3, 1], anchors[4, 1])
alignedDNA2 <- AlignProfiles(dna1, dna2, anchor=anchors)
```

AlignSeqs *Align a Set of Unaligned Sequences*

Description

Performs profile-to-profile alignment of multiple unaligned sequences following a guide tree.

Usage

```
AlignSeqs(myXStringSet,
          guideTree = NULL,
          iterations = 2,
          refinements = 1,
          gapOpening = c(-18, -10),
          gapExtension = -3,
          useStructures = TRUE,
```

```
          structures = NULL,
          FUN = AdjustAlignment,
          levels = c(0.9, 0.7, 0.7, 0.4, 10, 5, 5, 2),
          alphabet = AA_REDUCED[[1]],
          processors = 1,
          verbose = TRUE,
          ...)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of unaligned sequences.
guideTree	Either NULL or a dendrogram giving the ordered tree structure in which to align profiles. If NULL then a guide tree will be automatically constructed based on the order of shared k-mers.
iterations	Number of additional realignment iterations to perform. During each iteration step the guideTree is regenerated based on the alignment and the sequences are realigned.
refinements	Number of refinement steps to perform. During each refinement step groups of sequences are realigned to rest of the sequences, and the best of these two alignments (before and after realignment) is kept.
gapOpening	Single numeric giving the cost for opening a gap in the alignment, or two numbers giving the minimum and maximum costs. In the latter case the cost will be varied depending upon whether the groups of sequences being aligned are nearly identical or maximally distant.
gapExtension	Single numeric giving the cost for extending an open gap in the alignment, or two numbers giving the minimum and maximum costs. In the latter case the cost will be varied depending upon whether the groups of sequences being aligned are nearly identical or maximally distant.
useStructures	Logical indicating whether to use secondary structure predictions during alignment. If TRUE (the default), secondary structure probabilities will be automatically calculated for amino acid and RNA sequences if they are not provided (i.e., when structures is NULL).
structures	Either a list of secondary structure probabilities matching the structureMatrix, such as that output by PredictHEC or PredictDBN, or NULL to generate the structures automatically. Only applicable if myXStringSet is an AAStringSet or RNAStringSet.
FUN	A function to be applied after each profile-to-profile alignment. (See details section below.)
levels	Numeric with eight elements specifying the levels at which to trigger events. (See details section below.)
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between sequences. A non-reduced amino acid alphabet can be used by setting alphabet equal to AA_STANDARD. Only applicable if myXStringSet is an AAStringSet.

processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Further arguments to be passed directly to AlignProfiles, including perfectMatch, misMatch, gapPower, terminalGap, restrict, anchor, normPower, standardize, substitutionMatrix, and structureMatrix.

Details

The profile-to-profile method aligns a sequence set by merging profiles along a guide tree until all the input sequences are aligned. This process has three main steps: (1) If guideTree=NULL, an initial single-linkage guide tree is constructed based on a distance matrix of shared k-mers. Alternatively, a dendrogram object can be provided as the initial guideTree. (2) If iterations is greater than zero, then a UPGMA guide tree is built based on the initial alignment and the sequences are re-aligned along this tree. This process repeated iterations times or until convergence. (3) If refinements is greater than zero, then subsets of the alignment are re-aligned to the remainder of the alignment. This process generates two alignments, the best of which is chosen based on its sum-of-pairs score. This refinement process is repeated refinements times, or until convergence.

The purpose of levels is to speed-up the alignment process by not running time consuming processes when they are unlikely to change the outcome. The first four levels control when refinements occur and the function FUN is run on the alignment. The default levels specify that these events should happen when above 0.9 (AA; levels[1]) or 0.7 (DNA/RNA; levels[3]) average dissimilarity on the initial tree, when above 0.7 (AA; levels[2]) or 0.4 (DNA/RNA; levels[4]) average dissimilarity on the iterative tree(s), and after every tenth improvement made during refinement. The sixth element of levels (levels[6]) prevents FUN from being applied at any point to less than 5 sequences.

The FUN function is always applied before returning the alignment so long as there are at least levels[6] sequences. The default FUN is AdjustAlignment, but FUN can be any function that takes in an XStringSet as its first argument, as well as weights, processors, and substitutionMatrix as optional arguments. For example, the default FUN could be altered to not perform any changes by setting it equal to function(x, ...) return(x), where x is an XStringSet.

Secondary structures are automatically computed for amino acid and RNA sequences unless structures are provided or useStructures is FALSE. Use of structures generally provides a moderate improvement in average accuracy on difficult-to-align sequences. The default structureMatrix is used unless an alternative is provided. For RNA sequences, consensus secondary structures are only computed when the total length of the initial guide tree is at least 5 (levels[7]) or the length of subsequent trees is at least 2 (levels[8]). Note that input RNAStringSets are assumed to be structured non-coding RNAs. Largely unstructured RNAs should be aligned with useStructures set to FALSE or, ideally, aligned with AlignTranslation if coding sequences (i.e., mRNAs).

Value

An XStringSet of aligned sequences.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics, 16, 322. http://doi.org/10.1186/s12859-015-0749-z

Wright, E. S. (2020). RNAconTest: comparing tools for noncoding RNA multiple sequence alignment based on structural consistency. RNA 2020, 26, 531-540.

See Also

AdjustAlignment, AlignDB, AlignProfiles, AlignSynteny, AlignTranslation, ReadDendrogram, Treeline, StaggerAlignment

Run `vignette("ArtOfAlignmentInR", package = "DECIPHER")` to see a related vignette.

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna <- RemoveGaps(dna)
alignedDNA <- AlignSeqs(dna)
BrowseSeqs(alignedDNA, highlight=1)

## Not run:
# use secondary structure with RNA sequences
alignedRNA <- AlignSeqs(RNAStringSet(dna))
BrowseSeqs(alignedRNA, highlight=1)

## End(Not run)
```

`AlignSynteny` *Pairwise Aligns Syntenic Blocks*

Description

Performs pairwise alignment of all blocks of synteny between sets of sequences.

Usage

```
AlignSynteny(synteny,
             dbFile,
             tblName = "Seqs",
             identifier = "",
             processors = 1,
             verbose = TRUE,
             ...)
```

Arguments

synteny	An object of class "Synteny".
dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the sequences are located that were used to create the object synteny.
identifier	Optional character string used to narrow the search results to those matching a specific identifier, or an integer sequence corresponding to indices of rownames(synteny). If "" (the default), then all identifiers are selected from synteny.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Further arguments to be passed directly to AlignProfiles, including perfectMatch, misMatch, gapPower, terminalGap, restrict, normPower, and substitutionMatrix.

Details

AlignSynteny will extract all sequence regions belonging to syntenic blocks in the input synteny object, and perform pairwise alignment with AlignProfiles. Hits are used to anchor the alignment such that only the regions between anchors are aligned.

Value

A list with elements for each pair of identifiers in synteny. Each list element contains a DNAStringSetList one pairwise alignment per syntenic block.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

FindSynteny, Synteny-class

Run vignette("ArtOfAlignmentInR", package = "DECIPHER") to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
 synteny <- FindSynteny(db, minScore=50)
 DNA <- AlignSynteny(synteny, db)
 names(DNA)
 DNA[[1]] # the first set of pairwise alignments
 DNA[[1]][[1]] # the first block of synteny between H9N2 & H5N1
 unlist(DNA[[2]]) # a DNAStringSet of synteny between H9N2 & H2N2
}
```

AlignTranslation *Align Sequences By Their Amino Acid Translation*

Description

Performs alignment of a set of DNA or RNA sequences by aligning their corresponding amino acid sequences.

Usage

```
AlignTranslation(myXStringSet,
                 sense = "+",
                 direction = "5' to 3'",
                 readingFrame = NA,
                 type = class(myXStringSet),
                 geneticCode = GENETIC_CODE,
                 ...)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet object of unaligned sequences.
sense	Single character specifying sense of the input sequences, either the positive ("+") coding strand or negative ("-") non-coding strand.
direction	Direction of the input sequences, either "5' to 3'" or "3' to 5'".
readingFrame	Numeric vector giving a single reading frame for all of the sequences, or an individual reading frame for each sequence in myXStringSet. The readingFrame can be either 1, 2, 3 to begin translating on the first, second, and third nucleotide position, or NA (the default) to guess the reading frame. (See details section below.)
type	Character string indicating the type of output desired. This should be one of "DNAStringSet", "RNAStringSet", "AAStringSet", or "both". (See value section below.)
geneticCode	Either a character vector giving the genetic code in the same format as GENETIC_CODE (the default), or a list containing one genetic code for each sequence in myXStringSet.
...	Further arguments to be passed directly to AlignSeqs, including gapOpening, gapExtension, gapPower, terminalGap, restrict, anchor, normPower, substitutionMatrix, structureMatrix, standardize, alphabet, guideTree, iterations, refinements, useStructures, structures, FUN, and levels.

Details

Alignment of proteins is often more accurate than alignment of their coding nucleic acid sequences. This function aligns the input nucleic acid sequences via aligning their translated amino acid sequences. First, the input sequences are translated according to the specified sense, direction, and readingFrame. The resulting amino acid sequences are aligned using AlignSeqs, and this alignment is (conceptually) reverse translated into the original sequence type, sense, and direction.

Not only is alignment of protein sequences generally more accurate, but aligning translations will ensure that the reading frame is maintained in the nucleotide sequences.

If the readingFrame is NA (the default) then an attempt is made to guess the reading frame of each sequence based on the number of stop codons in the translated amino acids. For each sequence, the first reading frame without stop codons will be chosen (either 1, 2, or 3), excluding stop codons in the final position. If the number of stop codons is inconclusive for a sequence then the reading frame will default to 1. The entire length of each sequence is translated in spite of any stop codons identified. Note that automatic readingFrame selection is only reliable in circumstances where there is a substantially long coding sequence with at most a single stop codon expected in the final position, and therefore it is preferable to specify the reading frame of each sequence if it is known.

Value

An XStringSet of the class specified by type, or a list of two components (nucleotides and amino acids) if type is "both". Note that incomplete starting and ending codons will be translated into the mask character ("+") if the result includes an AAStringSet.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2015). DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. BMC Bioinformatics, 16, 322. http://doi.org/10.1186/s12859-015-0749-z

See Also

AlignDB, AlignProfiles, AlignSeqs, AlignSynteny, CorrectFrameshifts

Run vignette("ArtOfAlignmentInR", package = "DECIPHER") to see a related vignette.

Examples

```
# first three sequences translate to  MGFITP*
# and the last sequence translates as MGF-TP*
rna <- RNAStringSet(c("AUGGGUUUCAUCACCCCCUAA", "AUGGGCUUCAUAACUCCUUGA",
 "AUGGGAUUCAUUACACCGUAG", "AUGGGGUUUACCCCAUAA"))
RNA <- AlignSeqs(rna, verbose=FALSE)
RNA

RNA <- AlignTranslation(rna, verbose=FALSE)
RNA

AA <- AlignTranslation(rna, type="AAStringSet", verbose=FALSE)
AA

BOTH <- AlignTranslation(rna, type="both", verbose=FALSE)
BOTH

# example of aligning many protein coding sequences:
```

```
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
DNA <- AlignTranslation(dna) # align the translation then reverse translate
DNA

# using a mixture of standard and non-standard genetic codes
gC1 <- getGeneticCode(id_or_name2="1", full.search=FALSE, as.data.frame=FALSE)
# Mollicutes use an alternative genetic code
gC2 <- getGeneticCode(id_or_name2="4", full.search=FALSE, as.data.frame=FALSE)
w <- grep("Mycoplasma|Ureaplasma", names(dna))
gC <- vector("list", length(dna))
gC[-w] <- list(gC1)
gC[w] <- list(gC2)
AA <- AlignTranslation(dna, geneticCode=gC, type="AAStringSet")
BrowseSeqs(AA)
```

AmplifyDNA *Simulate Amplification of DNA by PCR*

Description

Predicts the amplification efficiency of theoretical PCR products (amplicons) given one or more primer sequences.

Usage

```
AmplifyDNA(primers,
        myDNAStringSet,
        maxProductSize,
        annealingTemp,
        P,
        ions = 0.2,
        includePrimers=TRUE,
        minEfficiency = 0.001,
        ...)
```

Arguments

primers	A DNAStringSet object or character vector with one or more unaligned primer sequences in 5' to 3' orientation.
myDNAStringSet	A DNAStringSet object or character vector with unaligned template DNA sequences in 5' to 3' orientation.
maxProductSize	Integer specifying the maximum length of PCR products (amplicons) in nucleotides.
annealingTemp	Numeric specifying the annealing temperature used in the PCR reaction.
P	Numeric giving the molar concentration of primers in the reaction.

ions	Numeric giving the molar sodium equivalent ionic concentration. Values may range between 0.01M and 1M.
includePrimers	Logical indicating whether to include the primer sequences in the theoretical PCR products. (See details section below.)
minEfficiency	Numeric giving the minimum amplification efficiency of PCR products to include in the output (default 0.1%). (See details section below.)
...	Additional arguments to be passed directly to CalculateEfficiencyPCR, including batchSize, taqEfficiency, maxDistance, maxGaps, and processors.

Details

Exponential amplification in PCR requires the annealing and elongation of two primers from target sites on opposing strands of the template DNA. If the template DNA sequence (e.g., chromosome) is known then predictions of theoretical amplicons can be obtained from in silico simulations of amplification. AmplifyDNA first searches for primer target sites on the template DNA, and then calculates an amplification efficiency from each target site using CalculateEfficiencyPCR. Ambiguity codes (IUPAC_CODE_MAP) are supported in the primers, but not in myDNAStringSet to prevent trivial matches (e.g., runs of N's).

If taqEfficiency is TRUE (the default), the amplification efficiency of each primer is defined as the product of hybridization efficiency and elongation efficiency. Amplification efficiency must be at least minEfficiency for a primer to be amplified in silico. Overall amplification efficiency of the PCR product is then calculated as the geometric mean of the two (i.e., forward and reverse) primers' efficiencies. Finally, amplicons are generated if the two primers are within maxProductSize nucleotides downstream of each other.

Potential PCR products are returned, either with or without including the primer sequences in the amplicon. The default (includePrimers=TRUE) is to incorporate the primer sequences as would normally occur during amplification. The alternative is to return the complete template sequence including the target sites, which may not exactly match the primer sequences. Note that amplicons may be duplicated when different input primers can amplify the same region of DNA.

Value

A DNAStringSet object containing potential PCR products, sorted from highest-to-lowest amplification efficiency. The sequences are named by their predicted amplification efficiency followed by the index of each primer in primers and the name (or index if names are missing) of the amplified sequence in myDNAStringSet. (See examples section below.)

Note

The program OligoArrayAux (http://www.unafold.org/Dinamelt/software/oligoarrayaux.php) must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the R console:

```
system("hybrid-min -V")
```

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2013) "Exploiting Extension Bias in PCR to Improve Primer Specificity in Ensembles of Nearly Identical DNA Templates." Environmental Microbiology, doi:10.1111/1462-2920.12259.

See Also

CalculateEfficiencyPCR, DesignPrimers, DesignSignatures, MeltDNA

Examples

```
data(yeastSEQCHR1)

# not run (must have OligoArrayAux installed first):

# match a single primer that acts as both the forward and reverse
primer1 <- "TGGAAGCTGAAACG"
## Not run: AmplifyDNA(primer1, yeastSEQCHR1, annealingTemp=55, P=4e-7, maxProductSize=500)

# perform a typical amplification with two primer sequences:
primer2 <- c("GGCTGTTGTTGGTGTT", "TGTCATCAGAACACCAA")
## Not run: AmplifyDNA(primer2, yeastSEQCHR1, annealingTemp=55, P=4e-7, maxProductSize=500)

# perform a multiplex PCR amplification with multiple primers:
primers <- c(primer1, primer2)
## Not run: AmplifyDNA(primers, yeastSEQCHR1, annealingTemp=55, P=4e-7, maxProductSize=500)
```

Array2Matrix *Create a Matrix Representation of a Microarray*

Description

Converts the output of DesignArray into the sparse matrix format.

Usage

```
Array2Matrix(probes,
             verbose = TRUE)
```

Arguments

probes	A set of microarray probes in the format output by DesignArray.
verbose	Logical indicating whether to display progress.

Details

A microarray can be represented by a matrix of hybridization efficiencies, where the rows represent each of the probes and the columns represent each the possible templates. This matrix is sparse since microarray probes are designed to only target a small subset of the possible templates.

Value

A list specifying the hybridization efficiency of each probe to its potential templates.

i	Element's row index in the sparse matrix.
j	Element's column index in the sparse matrix.
x	Non-zero elements' values representing hybridization efficiencies.
dimnames	A list of two components: the names of each probe, and the names of each template.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2013) Identification of Bacterial and Archaeal Communities From Source to Tap. Water Research Foundation, Denver, CO.

DR Noguera, et al. (2014). Mathematical tools to optimize the design of oligonucleotide probes and primers. Applied Microbiology and Biotechnology. doi:10.1007/s00253-014-6165-x.

See Also

DesignArray, NNLS

Run vignette("DesignMicroarray", package = "DECIPHER") to see a related vignette.

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
names(dna) <- 1:length(dna)
probes <- DesignArray(dna)
A <- Array2Matrix(probes)
```

BLOSUM *BLOSUM Amino Acid Substitution Matrices*

Description

The BLOSUM amino acid substitution matrices defined by Henikoff, S., & Henikoff, J. (1992).

Usage

```
data("BLOSUM")
```

Format

The format is: num [1:24, 1:24, 1:15] 2.4 -0.6 0 0 -1.8 0.6 0 0 -1.2 0 ... - attr(*, "dimnames")=List of 3 ..$: chr [1:24] "A" "R" "N" "D" ... ..$: chr [1:24] "A" "R" "N" "D" ... ..$: chr [1:15] "30" "35" "40" "45" ...

Details

Substitution matrix values represent the log-odds of observing an aligned pair of amino acids versus the likelihood of finding the pair by chance. The PFASUM substitution matrices are stored as an array named by each sub-matrix's similarity threshold. (See examples section below.) In all cases values are in units of third-bits ($log(odds\ ratio) * 3/log(2)$).

Source

Henikoff, S., & Henikoff, J. (1992). Amino Acid Substitution Matrices from Protein Blocks. *PNAS*, **89(22)**, 10915-10919.

Examples

```
data(BLOSUM)
BLOSUM62 <- BLOSUM[,, "62"] # the BLOSUM62 matrix
BLOSUM62["A", "R"] # score for A/R pairing

data(PFASUM)
plot(PFASUM[1:20, 1:20, "62"], BLOSUM62[1:20, 1:20])
abline(a=0, b=1)
```

BrowseDB *View a Database Table in a Web Browser*

Description

Opens an html file in a web browser to show the contents of a table in a database.

Usage

```
BrowseDB(dbFile,
         htmlFile = tempfile(fileext=".html"),
         openURL = interactive(),
         tblName = "Seqs",
         identifier = "",
         limit = -1,
         orderBy = "row_names",
         maxChars = 50,
         title = "",
         clause="")
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
htmlFile	Character string giving the location where the html file should be written.
openURL	Logical indicating whether the htmlFile should be opened in a web browser.
tblName	Character string specifying the table to view.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected.
limit	Number of results to display. The default (-1) does not limit the number of results.
orderBy	Character string giving the column name for sorting the results. Defaults to the order of entries in the database. Optionally can be followed by " ASC" or " DESC" to specify ascending (the default) or descending order.
maxChars	Maximum number of characters to display in each column.
title	Character string denoting a title that should appear at the top of the output or "" (the default) for no title.
clause	An optional character string to append to the query as part of a "where clause".

Value

Creates an html table containing all the fields of the database table and (if openURL is TRUE) opens it in the web browser for viewing.

Returns htmlFile if the html file was written successfully.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

See Also

BrowseSeqs

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 BrowseDB(db)
}
```

BrowseSeqs *View Sequences in a Web Browser*

Description

Opens an html file in a web browser to show the sequences in an XStringSet.

Usage

```
BrowseSeqs(myXStringSet,
           htmlFile = tempfile(fileext=".html"),
           openURL = interactive(),
           colorPatterns = TRUE,
           highlight = NA,
           patterns = c("-", alphabet(myXStringSet, baseOnly=TRUE)),
           colors = substring(rainbow(length(patterns),
                              v=0.8, start=0.9, end=0.7), 1, 7),
           fontColor="white",
           colWidth = Inf,
           title = "",
           ...)
```

Arguments

myXStringSet	A XStringSet object of sequences.
htmlFile	Character string giving the location where the html file should be written.
openURL	Logical indicating whether the htmlFile should be opened in a web browser.
colorPatterns	Logical specifying whether to color the background of matching patterns, or an integer vector providing pairs of start and stop boundaries for coloring.
highlight	Numeric specifying which sequence in the set to use for comparison or NA to color all sequences (default). If highlight is 0 then positions differing from the consensus sequence are highlighted.
patterns	An AAStringSet, DNAStringSet, or RNAStringSet object, a character vector containing regular expressions, a list of numeric matrices, or NULL. (See details section below.)
colors	Character vector providing the color for each of the matched patterns. Typically a character vector with elements of 7 characters: "#" followed by the red, blue, green values in hexadecimal (after rescaling to 0 ... 255). Ignored when patterns is a list of matrices.
fontColor	Character string denoting the font color for sequence text with a colored background, or a character vector with a color for each element of colors.
colWidth	Integer giving the maximum number of nucleotides wide the display can be before starting a new page. Must be a multiple of 20 (e.g., 100), or Inf (the default) to display all the sequences in one set of rows.

title	Character string denoting a title that should appear at the top of the output or "" (the default) for no title.
...	Additional arguments to adjust the appearance of the consensus sequence at the base of the display. Passed directly to ConsensusSequence for an AAStringSet, DNAStringSet, or RNAStringSet, or to consensusString for a BStringSet.

Details

BrowseSeqs converts an XStringSet into html format for viewing in a web browser. The sequences are colored in accordance with the patterns that are provided, or left uncolored if colorPatterns is FALSE or patterns is NULL. Character or XStringSet patterns are matched as regular expressions and colored according to colors. If patterns is a list of matrices, then it must contain one element per sequence. Each matrix is interpreted as providing the fraction red, blue, and green for each letter in the sequence. Thus, colors is ignored when patterns is a list. (See examples section below.)

Patterns are not matched across column breaks, so multi-character patterns should be carefully considered when colWidth is less than the maximum sequence length. Patterns are matched sequentially in the order provided, so it is feasible to use nested patterns such as c("ACCTG", "CC"). In this case the "CC" could be colored differently inside the previously colored "ACCTG". Note that patterns overlapping the boundaries of a previously matched pattern will not be matched. For example, "ACCTG" would not be matched if patterns=c("CC", "ACCTG").

Some web browsers cannot quickly display a large amount colored text, so it is recommended to use colorPatterns = FALSE or to highlight a sequence when viewing a large XStringSet. Highlighting will only show all of the characters in the highlighted sequence, and convert all matching positions in the other sequences into dots without color. Also, note that some web browsers display small shifts between fixed-width characters that may become noticeable as color offsets between the ends of long sequences.

Value

Creates an html file containing sequence data and (if openURL is TRUE) opens it in a web browser for viewing. The layout has the sequence name on the left, position legend on the top, cumulative number of nucleotides on the right, and consensus sequence on the bottom.

Returns htmlFile if the html file was written successfully.

Note

Some web browsers do not display colored characters with equal widths. If positions do not align across sequences then try opening the htmlFile with a different web browser.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

Kunzmann P., et al. (2020) "Substitution matrix based color schemes for sequence alignment visualization". BMC Bioinformatics, **21(1):209**.

See Also

BrowseDB, ConsensusSequence

Examples

```
# load the example DNA sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas) # non-coding ribosomal RNA gene sequences

# example of using the defaults with DNA sequences
BrowseSeqs(dna) # view the XStringSet

# color only "ACTG" and "CSC" patterns (where S is C or G)
BrowseSeqs(dna, patterns=DNAStringSet(c("ACTG", "CSC")))

# highlight (i.e., only fully-color) the first sequence
BrowseSeqs(dna, highlight=1) # other sequences are dots where matching

# highlight the consensus sequence at the bottom
BrowseSeqs(dna, highlight=0) # other sequences are dots where matching

# split the wide view into multiple vertical pages (for printing)
BrowseSeqs(dna, colWidth=100, highlight=1)

# specify an alternative color scheme for -, A, C, G, T
BrowseSeqs(dna, colors=c("#1E90FF", "#32CD32", "#9400D3", "black", "#EE3300"))

# only color the positions within certain positional ranges (100-200 & 250-500)
BrowseSeqs(dna, colorPatterns=c(100, 200, 250, 500))

# example of calling attention to letters by coloring gaps black
BrowseSeqs(dna, patterns="-", colors="black")

# add a title to the top of the page
BrowseSeqs(dna, title="Title")

## Not run:
# color according to base-pairing by supplying the fraction RGB for every position
dbn <- PredictDBN(dna, type="structures") # calculate the secondary structures
# dbn now contains the scores for whether a base is paired (left/right) or unpaired
dbn[[1]][, 1] # the scores for the first position in the first sequence
dbn[[2]][, 10] # the scores for the tenth position in the second sequence
# these positional scores can be used as shades of red, green, and blue:
BrowseSeqs(dna, patterns=dbn) # red = unpaired, green = left-pairing, blue = right
# positions in black are not part of the consensus secondary structure

## End(Not run)

# color all restriction sites
data(RESTRICTION_ENZYMES) # load dataset containing restriction enzyme sequences
sites <- RESTRICTION_ENZYMES
sites <- gsub("[^A-Z]", "", sites) # remove non-letters
```

```
sites <- DNAStringSet(sites) # convert the character vector to a DNAStringSet
rc_sites <- reverseComplement(DNAStringSet(sites))
w <- which(sites != rc_sites) # find non-palindromic restriction sites
sites <- c(sites, rc_sites[w]) # append their reverse complement
sites <- sites[order(nchar(sites))] # match shorter sites first
BrowseSeqs(dna, patterns=sites)

# color bases by quality score
fastq <- system.file("extdata", "s_1_sequence.txt", package="Biostrings")
reads <- readQualityScaledDNAStringSet(fastq, quality.scoring="solexa")
colors <- colorRampPalette(c("red", "yellow", "green"))(42)
colors <- col2rgb(colors)/255
quals <- as(quality(reads), "IntegerList")
quals <- lapply(quals, function(x) colors[, x])
BrowseSeqs(DNAStringSet(reads), patterns=quals) # green = high quality, red = low quality

# load the example protein coding sequences
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

# example of using the defaults with amino acid sequences
aa <- unique(translate(dna)) # the unique amino acid sequences
BrowseSeqs(aa)

# example of highlighting the consensus amino acid sequence
AA <- AlignSeqs(aa)
BrowseSeqs(AA, highlight=0)

# example of highlighting positions that differ from the majority consensus
BrowseSeqs(AA, highlight=0, threshold=0.5)

# specify an alternative color scheme for amino acids (from Kunzmann et al.)
colors <- c(`-`="#000000", `A`="#BDB1E8", `R`="#EFA2C5", `N`="#F6602F",
    `D`="#FD5559", `C`="#12C7FE", `Q`="#DDACB4", `E`="#FEA097", `G`="#F46802",
    `H`="#FCA708", `I`="#369BD9", `L`="#2E95EC", `K`="#CF7690", `M`="#4B8EFE",
    `F`="#76997D", `P`="#FD2AE3", `S`="#A08A9A", `T`="#9A84D5", `W`="#74C80D",
    `Y`="#9BB896", `V`="#89B9F9")
BrowseSeqs(AA, colors=colors, patterns=names(colors))

# example of coloring in a reduced amino acid alphabet
alpha <- AA_REDUCED[[15]]
alpha # clustering of amino acids based on similarity
BrowseSeqs(AA, patterns=c("-", paste("[", alpha, "]", sep="")))

# color amino acids according to their predicted secondary structure
hec <- PredictHEC(AA, type="probabilities") # calculate the secondary structures
# hec now contains the probability that a base is in an alpha-helix or beta-sheet
hec[[3]][, 18] # the 18th position in sequence 3 is likely part of a beta-sheet (E)
# the positional probabilities can be used as shades of red, green, and blue:
BrowseSeqs(AA, patterns=hec) # red = alpha-helix, green = beta-sheet, blue = coil

# color codons according to their corresponding amino acid
DNA <- AlignTranslation(dna) # align the translation then reverse translate
```

```
colors <- rainbow(21, v=0.8, start=0.9, end=0.7) # codon colors
m <- match(GENETIC_CODE, unique(GENETIC_CODE)) # corresponding amino acid
codonBounds <- matrix(c(seq(1, width(DNA)[1], 3), # start of codons
 seq(3, width(DNA)[1], 3)), # end of codons
 nrow=2,
 byrow=TRUE)
BrowseSeqs(DNA,
 colorPatterns=codonBounds,
 patterns=c("---", names(GENETIC_CODE)), # codons to color
 colors=c("black", substring(colors[m], 1, 7)))
```

`CalculateEfficiencyArray` *Predict the Hybridization Efficiency of Probe/Target Sequence Pairs*

Description

Calculates the Gibbs free energy and hybridization efficiency of probe/target pairs at varying concentrations of the denaturant formamide.

Usage

```
CalculateEfficiencyArray(probe,
                         target,
                         FA = 0,
                         dGini = 1.96,
                         Po = 10^-2.0021,
                         m = 0.1731,
                         temp = 42,
                         deltaGrules = NULL)
```

Arguments

`probe`	A `DNAStringSet` object or character vector with pairwise-aligned probe sequences in 5' to 3' orientation.
`target`	A `DNAStringSet` object or character vector with pairwise-aligned target sequences in 5' to 3' orientation.
`FA`	A vector of one or more formamide concentrations (as percent v/v).
`dGini`	The initiation free energy. The default is 1.96 [kcal/mol].
`Po`	The effective probe concentration.
`m`	The m-value defining the linear relationship of denaturation in the presence of formamide.
`temp`	Equilibrium temperature in degrees Celsius.
`deltaGrules`	Free energy rules for all possible base pairings in quadruplets. If NULL, defaults to the parameters obtained using NimbleGen microarrays and a Linear Free Energy Model developed by Yilmaz *et al*.

Details

This function calculates the free energy and hybridization efficiency (HE) for a given formamide concentration ([FA]) using the linear free energy model given by:

$$HE = Po * exp[-(dG_0 + m * FA)/RT]/(1 + Po * exp[-(dG_0 + m * FA)/RT])$$

The `probe` and `target` input sequences must be aligned in pairs, such that the first probe is aligned to the first target, second-to-second, and so on. Ambiguity codes in the `IUPAC_CODE_MAP` are accepted in probe and target sequences. Any ambiguities will default to perfect match pairings by inheriting the nucleotide in the same position on the opposite sequence whenever possible. If the ambiguity results in a mismatch then "T", "G", "C", and "A" are substituted, in that order. For example, if a probe nucleotide is "S" ("C" or "G") then it will be considered a "C" if the target nucleotide in the same position is a "C", otherwise the ambiguity will be interpreted as a "G".

If `deltaGrules` is NULL then the rules defined in `data(deltaGrules)` will be used. Note that `deltaGrules` of the same format may be customized for any application and specified as an input.

Value

A `matrix` with the predicted Gibbs free energy (dG) and hybridization efficiency (HE) at each concentration of formamide ([FA]).

Author(s)

Erik Wright `<Erik.Wright@bcm.edu>`

References

Yilmaz LS, Loy A, Wright ES, Wagner M, Noguera DR (2012) Modeling Formamide Denaturation of Probe-Target Hybrids for Improved Microarray Probe Design in Microbial Diagnostics. PLoS ONE 7(8): e43862. doi:10.1371/journal.pone.0043862.

See Also

`deltaGrules`

Examples

```
probes <- c("AAAAACGGGGAGCGGGGGGATACTG", "AAAAACTCAACCCGAGGAGCGGGGG")
targets <- c("CAACCCGGGGAGCGGGGGGATACTG", "TCGGGCTCAACCCGAGGAGCGGGGG")
result <- CalculateEfficiencyArray(probes, targets, FA=0:40)
dG0 <- result[, "dG_0"]
HE0 <- result[, "HybEff_0"]
plot(result[1, 1:40], xlab="[FA]", ylab="HE", main="Probe/Target # 1", type="l")
```

CalculateEfficiencyFISH

Predict Thermodynamic Parameters of Probe/Target Sequence Pairs

Description

Calculates the Gibbs free energy, formamide melt point, and hybridization efficiency of probe/target (DNA/RNA) pairs.

Usage

```
CalculateEfficiencyFISH(probe,
                        target,
                        temp,
                        P,
                        ions,
                        FA,
                        batchSize = 1000)
```

Arguments

probe	A DNAStringSet object or character vector with unaligned probe sequences in 5' to 3' orientation.
target	A DNAStringSet object, RNAStringSet, or character vector with unaligned target or non-target sequences in 5' to 3' orientation. The DNA base Thymine will be treated the same as Uracil.
temp	Numeric specifying the hybridization temperature, typically 46 degrees Celsius.
P	Numeric giving the molar concentration of probes during hybridization.
ions	Numeric giving the molar sodium equivalent ionic concentration. Values may range between 0.01M and 1M. Note that salt correction is not available for thermodynamic rules of RNA/RNA interactions, which were determined at 1 molar concentration.
FA	Numeric concentration (as percent v/v) of the denaturant formamide in the hybridization buffer.
batchSize	Integer specifying the number of probes to simulate hybridization per batch. See the Description section below.

Details

Hybridization of pairwise probe/target (DNA/RNA) pairs is simulated *in silico*. Gibbs free energies are obtained from system calls to OligoArrayAux, which must be properly installed (see the Notes section below). Probe/target pairs are sent to OligoArrayAux in batches of batchSize, which prevents systems calls from being too many characters. Note that OligoArrayAux does not support degeneracy codes (non-base letters), although they are accepted without error. Any sequences with ambiguity should be expanded into multiple permutations with Disambiguate before input.

Value

A matrix of predicted hybridization efficiency (HybEff), formamide melt point (FAm), and free energy (ddG1 and dG1) for each probe/target pair of sequences.

Note

The program OligoArrayAux (http://www.unafold.org/Dinamelt/software/oligoarrayaux.php) must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the R console:

```
system("hybrid-min -V")
```

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2014) "Automated Design of Probes for rRNA-Targeted Fluorescence In Situ Hybridization Reveals the Advantages of Using Dual Probes for Accurate Identification." Applied and Environmental Microbiology, doi:10.1128/AEM.01685-14.

See Also

DesignProbes, TileSeqs

Examples

```
probe <- c("GGGCTTTCACATCAGACTTAAGAAACC", "CCCCACGCTTTCGCGCC")
target <- reverseComplement(DNAStringSet(probe))
# not run (must have OligoArrayAux installed first):
## Not run: CalculateEfficiencyFISH(probe, target, temp=46, P=250e-9, ions=1, FA=35)
```

CalculateEfficiencyPCR

Predict Amplification Efficiency of Primer Sequences

Description

Calculates the amplification efficiency of primers from their hybridization efficiency and elongation efficiency at the target site.

Usage

```
CalculateEfficiencyPCR(primer,
                       target,
                       temp,
                       P,
                       ions,
                       batchSize = 1000,
                       taqEfficiency = TRUE,
                       maxDistance = 0.4,
                       maxGaps = 2,
                       processors = 1)
```

Arguments

primer	A DNAStringSet object or character vector with unaligned primer sequences in 5' to 3' orientation.
target	A DNAStringSet object or character vector with unaligned target or non-target sequences in 5' to 3' orientation.
temp	Numeric specifying the annealing temperature used in the PCR reaction.
P	Numeric giving the molar concentration of primers in the reaction.
ions	Numeric giving the molar sodium equivalent ionic concentration. Values may range between 0.01M and 1M.
batchSize	Integer specifying the number of primers to simulate hybridization per batch. See the Description section below.
taqEfficiency	Logical determining whether to make use of elongation efficiency and maxDistance to increase predictive accuracy for *Taq* DNA Polymerase amplifying primers with mismatches near the 3' terminus. Note that this should be set to FALSE if using a high-fidelity polymerase with 3' to 5' exonuclease activity.
maxDistance	Numeric specifying the maximal fraction of mismatched base pairings on a rolling basis beginning from the 3' end of the primer. Only used if taqEfficiency is TRUE.
maxGaps	Integer specifying the maximum number of insertions or deletions (indels) in the primer/target alignment. Only used if taqEfficiency is TRUE.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.

Details

Amplification of pairwise primer/target pairs is simulated *in silico*. A complex model of hybridization is employed that takes into account the side reactions resulting from probe-folding, target-folding, and primer-dimer formation. The resulting hybridization efficiency is multiplied by the elongation efficiency to predict the overall efficiency of amplification.

Free energy is obtained from system calls to OligoArrayAux, which must be properly installed (see the Notes section below). Primer/target pairs are sent to OligoArrayAux in batches of batchSize, which prevents systems calls from being too many characters. Note that OligoArrayAux does not

support degeneracy codes (non-base letters), although they are accepted without error. Any sequences with ambiguity should be expanded into multiple permutations with Disambiguate before input.

Value

A vector of predicted efficiencies for amplifying each primer/target pair of sequences.

Note

The program OligoArrayAux (http://www.unafold.org/Dinamelt/software/oligoarrayaux.php) must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the R console:

```
system("hybrid-min -V")
```

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2013) "Exploiting Extension Bias in PCR to Improve Primer Specificity in Ensembles of Nearly Identical DNA Templates." Environmental Microbiology, doi:10.1111/1462-2920.12259.

See Also

AmplifyDNA, DesignPrimers, DesignSignatures

Examples

```
primers <- c("AAAAACGGGGAGCGGGGGG", "AAAAACTCAACCCGAGGAGCGCGT")
targets <- reverseComplement(DNAStringSet(primers))
# not run (must have OligoArrayAux installed first):
## Not run: CalculateEfficiencyPCR(primers, targets, temp=75, P=4e-7, ions=0.225)
```

Clusterize *Cluster Sequences By Distance*

Description

Groups the sequences into approximate clusters of similarity.

Usage

```
Clusterize(myXStringSet,
           cutoff = 0,
           method = "overlap",
           includeTerminalGaps = FALSE,
           penalizeGapLetterMatches = NA,
           minCoverage = 0.5,
           maxPhase1 = 2e4,
           maxPhase2 = 2e3,
           maxPhase3 = 2e3,
           maxAlignments = 200,
           rareKmers = 50,
           probability = 0.99,
           invertCenters = FALSE,
           singleLinkage = FALSE,
           maskRepeats = FALSE,
           maskLCRs = FALSE,
           alphabet = AA_REDUCED[[186]],
           processors = 1,
           verbose = TRUE,
           ...)
```

Arguments

myXStringSet	The (unaligned) DNAStringSet, RNAStringSet, or AAStringSet to cluster.
cutoff	A vector of maximum distances (approximately) separating sequences in the same cluster. Multiple cutoffs may be provided in ascending or descending order. (See details section below.)
method	Character string determining the region in which distance is calculated. This should be (an unambiguous abbreviation of) one of "overlap", "shortest", or "longest". The default method ("overlap") calculates distance from the overlapping region between terminal gaps when includeTerminalGaps is FALSE and the entire alignment otherwise. Setting method to "shortest" or "longest" will use the region between the start and end of the shortest or longest sequence, respectively, for each pairwise distance. The method is only applicable when includeTerminalGaps is TRUE.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" characters) in the pairwise alignments into the calculation of distance.
penalizeGapLetterMatches	Logical specifying whether or not to consider gap-to-letter matches as mismatches. If FALSE, then gap-to-letter matches are not included in the total length used to calculate distance, and if TRUE all gaps-to-letter pairs are considered mismatches. The default (NA) is to penalize gap-to-letter mismatches once per insertion or deletion, which treats runs of gaps (i.e., indels) as equivalent to a single mismatch.

minCoverage	Numeric giving the minimum fraction of sequence positions that must be overlapping for a sequence to be clustered with the cluster representative. If positive then coverage is calculated relative to the sequence being clustered (i.e., the shorter sequence). If negative then coverage is computed relative to the cluster representative (i.e., the longest sequence in the cluster).
maxPhase1	An integer specifying the maximum number of related sequences to consider in the initial partitioning of the sequences.
maxPhase2	An integer giving the maximum number of replicates to perform when sorting sequences based on their k-mer similarity.
maxPhase3	An integer determining the number of comparisons per sequence to perform when attempting to find cluster centers.
maxAlignments	An integer designating the maximum number of alignments to perform when attempting to assign a sequence to an existing cluster.
rareKmers	An integer setting the number of rare k-mers to record per sequence. Larger values require more memory but may improve accuracy with diminishing returns.
probability	Numeric between 0 and 1 (exclusive) defining the approximate probability of clustering sequences that are exactly cutoff distant. Typically near, but always less than, 1. Lower values result in faster clustering at the expense of effectiveness.
invertCenters	Logical controlling whether the cluster center is inverted (i.e., multiplied by -1), which allows the centers to be determined from the results. The default (FALSE) only returns positive cluster numbers. If TRUE, the center sequence(s) of each cluster are negative.
singleLinkage	Logical specifying whether to perform single-linkage clustering. The default (FALSE) only establishes linkage to the cluster center. Single-linkage clustering creates broader clusters that may better correspond to natural groups depending on the application.
maskRepeats	Logical specifying whether to mask repeats when clustering.
maskLCRs	Logical indicating whether to mask low complexity regions when clustering.
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between sequences.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Additional arguments to be passed directly to DistanceMatrix, including correction, substitutionMatrix, and frequencies.

Details

Clusterize groups the input sequences into approximate clusters using a heuristic algorithm with linear time and memory complexity. In phase 1, the sequences are partitioned into groups of similarity. In phase 2, the sequences are ordered by k-mer similarity by relatedness sorting. In phase 3, the sequences are iteratively clustered in this order by their similarity to surrounding sequences

in the sorting. That is, the first sequence becomes the representative of cluster #1. If the second sequence is within cutoff distance then it is added to the cluster, otherwise it becomes a new cluster representative. The remaining sequences are matched to cluster representatives in a similar fashion until all sequences belong to a cluster. In the majority of cases, this process results in clusters with members separated by less than cutoff distance, and all cluster members must be within cutoff distance of their cluster representative.

The calculation of distance can be controlled in multiple ways, with each parameterization of distance having advantages and disadvantages. By default, distance is the fraction of positions that are different, including gaps, within the overlapping region in a pairwise alignment. The defaults will handle partial-length sequences well, but also cluster sequences with high similarity between their opposite ends. For this reason, it is important to set minimumCoverage such that distances are based off of considerable overlap between sequences in the pairwise alignment. The default (0.5) requires sequences to share at least 50% of their positions with the cluster representative. This distance parameterization works well, but there are reasonable alternatives.

If penalizeGapLetterMatches is FALSE, the distance will exclude gap regions. If includeTerminalGaps is TRUE, the calculation of distance will use the entire (global) alignment. If method is "shortest" and includeTerminalGaps is set to TRUE, then the distance is calculated for the region encompassed by the shorter sequence in each pair, which is the common definition of distance used by other clustering programs. This common definition of distance will sometimes separate partly overlapping sequences, which is why it is not the default. Another option is to set minCoverage to a negative fraction. This requires sequences to have substantial overlap with the cluster representative, which is the longest sequence in the cluster. For example, setting minCoverage to -0.5 would require every clustered sequence to share at least 50% of positions with the cluster representative.

Distance can also be calculated under a model of evolution or based on the average (negative) substitution score, including any distance measure supported by DistanceMatrix. This can be accomplished by specifying correction, substitutionMatrix, and/or frequencies. Alternative distances can correct for unequal substitution rates and multiple substitutions per site, which provides greater resolution than the default (length-normalized Hamming) distance, especially for high values of cutoff. In these cases, cutoff is no longer a fraction, but becomes the distance in substitutions per site or units of the substitution matrix (depending on the parameterization). See the help page for DistanceMatrix to learn more about alternative distances.

The algorithm requires time proportional to the number of input sequences in myXStringSet. The phase 1, up to maxPhase1 sequences sharing a k-mer are tabulated while partitioning each sequence. In phase 2, the sequences are compared with up to maxPhase2 passes that each take linear time. Ordering of the sequences is performed in linear time using radix sorting. In phase 3, each sequence is compared with up to maxPhase3 previous cluster representatives of sequences sharing rareKmers or nearby sequences in the relatedness ordering. This is possible because the sequences are sorted by relatedness, such that more recent cluster representatives are more similar. Hence, the complete algorithm scales in linear time asymptotically and returns clusters of sequences within cutoff distance of their center sequence.

Multiple cutoffs can be provided in sorted order, which saves time because phases 1 and 2 only need to be performed once. If the cutoffs are provided in *descending* order then clustering at each new value of cutoff is continued within the prior cutoff's clusters. In this way clusters at lower values of cutoff are completely contained within their "umbrella" clusters at higher values of cutoff. This slightly accelerates the clustering process, because each subsequent group is only clustered within the previous group. If multiple cutoffs are provided in *ascending* order then clustering at each level of cutoff is independent of the prior level.

Note, the clustering algorithm is stochastic. Hence, clusters can vary from run-to-run unless the random number seed is set for repeatability (i.e., with `set.seed`). Also, `invertCenters` can be used to determine the center sequence of each cluster from the output. Since identical sequences will always be assigned the same cluster numbers, it is possible for more than one input sequence in `myXStringSet` to be assigned as the center of a cluster if they are identical.

Value

A data.frame is returned with dimensions $N * M$, where each one of N sequences is assigned to a cluster at the M-level of cutoff. The row.names of the data.frame correspond to the *names* of `myXStringSet`.

Note

Clusterize does not follow the convention of providing a similarity `cutoff` when clustering. This is because `cutoff` is defined as a distance, which might be greater than `1` when employing a model of evolution. Under the default settings, distance is a fraction and `cutoff` should be less than `1` (i.e., similarity above zero). However, `cutoff` is unbounded when providing argument(s) for ....

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2024). Accurately clustering biological sequences in linear time by relatedness sorting. Nature Communications, 15, 1-13. http://doi.org/10.1038/s41467-024-47371-9

See Also

`AA_REDUCED`, `DistanceMatrix`, `Treeline`

Run `vignette("ClusteringSequences", package = "DECIPHER")` to see a related vignette.

Examples

```
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
aa <- translate(dna)

# typical usage (e.g., clustering at >= 90 percent similarity)
clusters <- Clusterize(aa, cutoff=0.1) # set processors = NULL for max speed
head(clusters)

# typical usage (e.g., obtaining cluster representatives)
clusters <- Clusterize(aa, cutoff=0.1, invertCenters=TRUE)
aa[clusters[[1]] < 0]

# cluster each cutoff within the previous cluster (nested groups)
clusters <- Clusterize(aa, cutoff=seq(0.7, 0, -0.1))
head(clusters)
apply(clusters, 2, max) # number of clusters per cutoff
```

```
# cluster each cutoff independently (possibly fewer clusters per cutoff)
clusters <- Clusterize(aa, cutoff=seq(0, 0.7, 0.1))
head(clusters)
apply(clusters, 2, max) # number of clusters per cutoff

# make cluster center(s) negative for tracking
clusters <- Clusterize(aa, cutoff=0.5, invertCenters=TRUE)
head(clusters)
clusters[clusters$cluster < 0,, drop=FALSE]
unique(aa[clusters$cluster < 0]) # unique cluster centers
max(abs(clusters)) # number of clusters

# cluster nucleotide sequences
clusters <- Clusterize(dna, cutoff=0.2, invertCenters=TRUE)
head(clusters)
max(abs(clusters)) # number of clusters

# cluster DNA using a model of evolution for distance
clusters <- Clusterize(dna, cutoff=0.2, correction="TN93+F")
head(clusters)
max(clusters) # number of clusters

# cluster amino acids using a model of evolution for distance
clusters <- Clusterize(aa, cutoff=0.5, correction="WAG")
head(clusters)
max(clusters) # number of clusters
```

Codec *Compression/Decompression of Character Vectors*

Description

Compresses character vectors into raw vectors, or decompresses raw vectors into character vectors using a variety of codecs.

Usage

```
Codec(x,
      compression,
      compressRepeats = FALSE,
      processors = 1)
```

Arguments

x	Either a character vector to be compressed, or a list of raw vectors to be decompressed.

`compression` The type of compression algorithm to use when `x` is a character vector. This should be (an unambiguous abbreviation of) one of `"nbit"` (for nucleotides), `"qbit"` (for quality scores), `"gzip"`, `"bzip2"`, or `"xz"`. If `compression` is `"nbit"` or `"qbit"` then a second method can be provided for cases when `x` is incompressible. Decompression type is determined automatically. (See details section below.)

`compressRepeats`
Logical specifying whether to compress exact repeats and reverse complement repeats in a character vector input (`x`). Only applicable when `compression` is `"nbit"`. Repeat compression in long DNA sequences generally increases compression by about 2% while requiring three-fold more compression time.

`processors` The number of processors to use, or `NULL` to automatically detect and use all available processors.

Details

`Codec` can be used to compress/decompress character vectors using different algorithms. The `"nbit"` and `"qbit"` methods are tailored specifically to nucleotides and quality scores, respectively. These two methods will store the data as plain text (`"ASCII"` format) when it is incompressible. In such cases, a second `compression` method can be given to use in lieu of plain text. For example `compression = c("nbit", "gzip")` will use `"gzip"` compression when `"nbit"` compression is inappropriate.

When performing the reverse operation, decompression, the type of `compression` is automatically detected based on the unique signature ("magic number") added by each compression algorithm.

Value

If `x` is a character vector to be compressed, the output is a list with one element containing a raw vector per character string. If `x` is a list of raw vectors to be decompressed, then the output is a character vector with one string per list element.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- as.character(readDNAStringSet(fas)) # aligned sequences
object.size(dna)

# compression
system.time(x <- Codec(dna, compression="nbit"))
object.size(x)/sum(nchar(dna)) # bytes per position

system.time(g <- Codec(dna, compression="gzip"))
object.size(g)/sum(nchar(dna)) # bytes per position

# decompression
system.time(y <- Codec(x))
```

```
stopifnot(dna==y)

system.time(z <- Codec(g))
stopifnot(dna==z)
```

ConsensusSequence *Create a Consensus Sequence*

Description

Forms a consensus sequence representing a set of sequences.

Usage

```
ConsensusSequence(myXStringSet,
                  threshold = 0.05,
                  ambiguity = TRUE,
                  noConsensusChar = "+",
                  minInformation = 1 - threshold,
                  includeNonLetters = FALSE,
                  includeTerminalGaps = FALSE)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of aligned sequences.
threshold	Numeric specifying that less than threshold fraction of sequence information can be lost at any position of the consensus sequence.
ambiguity	Logical specifying whether to consider ambiguity as split between their respective nucleotides. Degeneracy codes are specified in the IUPAC_CODE_MAP.
noConsensusChar	Single character from the sequence's alphabet giving the base to use when there is no consensus in a position.
minInformation	Minimum fraction of information required to form consensus in each position.
includeNonLetters	Logical specifying whether to count gap ("-"), mask ("+"), and unknown (".") characters towards the consensus.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" or "." characters on each end of the sequence) into the formation of consensus.

Details

ConsensusSequence removes the least frequent characters at each position, so long as they represent less than threshold fraction of the sequences in total. If necessary, ConsensusSequence represents the remaining characters using a degeneracy code from the IUPAC_CODE_MAP. Degeneracy codes are always used in cases where multiple characters are equally abundant.

Two key parameters control the degree of consensus: `threshold` and `minInformation`. The default `threshold` (`0.05`) means that at less than 5% of sequences will not be represented by the consensus sequence at any given position. The default `minInformation` (`1 - 0.05`) specifies that at least 95% of sequences must contain the information in the consensus, otherwise the `noConsensusChar` is used. This enables an alternative character (e.g., "+") to be substituted at positions that would otherwise yield an ambiguity code.

If `ambiguity = TRUE` (the default) then degeneracy codes in `myXStringSet` are split between their respective bases according to the `IUPAC_CODE_MAP` for DNA/RNA and `AMINO_ACID_CODE` for AA. For example, an "R" in a `DNAStringSet` would count as half an "A" and half a "G". If `ambiguity = FALSE` then degeneracy codes are not considered in forming the consensus. For an `AAStringSet` input, the lack of degeneracy codes generally results in "X" at positions with mismatches, unless the `threshold` is set to a higher value than the default.

If `includeNonLetters = TRUE` (the default) then gap ("-"), mask ("+"), and unknown (".") characters are counted towards the consensus, otherwise they are omitted from calculation of the consensus. Note that gap ("-") and unknown (".") characters are treated interchangeably as gaps when forming the consensus sequence. For this reason, the consensus of a position with all unknown (".") characters will be a gap ("-"). Also, note that if consensus is formed between different length sequences then it will represent only the longest sequences at the end. For this reason the consensus sequence is generally based on a sequence alignment such that all of the sequences have equal lengths.

Value

An `XStringSet` with a single consensus sequence matching the input type.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

`Disambiguate`, `IdConsensus`

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas, n=10) # limit 10 sequences for visability
BrowseSeqs(dna) # consensus at bottom
BrowseSeqs(dna, threshold=0.5) # consensus at bottom

# controlling the degree of consensus
AAAT <- DNAStringSet(c("A", "A", "A", "T"))
ConsensusSequence(AAAT) # "W"
ConsensusSequence(AAAT, threshold=0.3) # "A"
ConsensusSequence(AAAT, threshold=0.3, minInformation=0.8) # "+"
ConsensusSequence(AAAT, threshold=0.3, minInformation=0.8, noConsensusChar="N") # "N"

# switch between degenerate-based and majority-based consensus
majority <- DNAStringSet(c("GTT", "GAA", "CTG"))
ConsensusSequence(majority) # degenerate-based
```

```
ConsensusSequence(majority, threshold=0.5) # majority-based
ConsensusSequence(majority, threshold=0.5, minInformation=0.75)

# behavior in the case of a tie
ConsensusSequence(DNAStringSet(c("A", "T"))) # "W"
ConsensusSequence(DNAStringSet(c("A", "T")), threshold=0.5) # "W"
ConsensusSequence(AAStringSet(c("A", "T"))) # "X"
ConsensusSequence(AAStringSet(c("A", "T")), threshold=0.5) # "X"
ConsensusSequence(AAStringSet(c("I", "L"))) # "J"
ConsensusSequence(AAStringSet(c("I", "L")), threshold=0.5) # "J"

# handling terminal gaps
dna <- DNAStringSet(c("ANGCT-","-ACCT-"))
ConsensusSequence(dna) # "ANSCT-"
ConsensusSequence(dna, includeTerminalGaps=TRUE) # "+NSCT-"

# the "." character is treated is a "-"
aa <- AAStringSet(c("ANQIH-", "ADELW."))
ConsensusSequence(aa) # "ABZJX-"

# internal non-bases are included by default
ConsensusSequence(DNAStringSet(c("A-+.A", "AAAAA")), noConsensusChar="N") # "ANNNA"
ConsensusSequence(DNAStringSet(c("A-+.A", "AAAAA")), includeNonLetters=TRUE) # "AAAAA"

# degeneracy codes in the input are considered by default
ConsensusSequence(DNAStringSet(c("AWNDA", "AAAAA"))) # "AWNDA"
ConsensusSequence(DNAStringSet(c("AWNDA", "AAAAA")), ambiguity=FALSE) # "AAAAA"
```

`Cophenetic` *Compute Cophenetic Distances on Dendrogram Objects*

Description

Calculates the matrix of cophenetic distances represented by a dendrogram object.

Usage

```
Cophenetic(x,
           distance = "length")
```

Arguments

`x`	A dendrogram object.
`distance`	The type of distance measure(s) to use. This should be `"length"` (the default), `"edges"`, and/or named edge attributes. Abbreviations are not permitted except in named edge attributes. (See details section below.)

Details

The cophenetic distance between two observations is defined by default as their separation on a dendrogram, also known as the patristic distance. This function differs from the `cophenetic` function in that it does not assume the tree is ultrametric and outputs (by default) the branch length separating pairs of observations (leaves) rather than the height of their merger.

The default distance is the path `length` between leaves of the dendrogram. Specifying `"edges"` will sum the number of edges separating leaves, while disregarding the root node of a bifurcating tree and any other edges without splits.

Alternatively, it is possible to customize the measure of distance by specifying a named edge attribute. The values associated with each attribute are converted to numbers and summed without `NA` values if containing more than one number. If an attribute is missing then it is ignored, with zero contribution to the distance.

If multiple `distance` measures are given, these are multiplied together (where they exist) along each edge to define more complex distance measures, such as the `"length"` times support probability of an edge. In this case, one `distance` must be `"length"` or `"edges"`, where the latter has no effect on distance (i.e., multiply by one).

Value

An object of class 'dist'.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

Treeline

Examples

```
# small example
x <- ReadDendrogram("(dog:8.125,(elephant:0,horse:7.5)0.5:4.375);")
plot(x) # star tree with "edgetext" on one branch
Cophenetic(x) # default distance = "length"
Cophenetic(x, distance="edges") # edge with root treated as one edge
Cophenetic(x, distance="edgetext") # missing attributes treated as zero
Cophenetic(x, distance=c("length", "edgetext")) # product of both per edge

# larger example
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
d1 <- DistanceMatrix(dna, type="dist", correction="JC69", verbose=FALSE)
dend <- Treeline(myDistMatrix=d1)
d2 <- Cophenetic(dend)
cor(d1, d2)

# examples of inter-node and inter-edge distances
d3 <- Cophenetic(dend, distance="edges")
```

```
cor(d2, d3) # perfect correlation
range(d3) # 2+ edges between leaves

# example of using an attribute as the distance
tree <- Treeline(dna,
 method="ML", # ML trees get aBayes support values
 model="JC69",
 maxTime=0) # set a stringent time cutoff for speed
attr(tree[[1]], "probability") # support value for first edge
d4 <- Cophenetic(tree, "probability")
cor(d1, d4)
# rescale aBayes (1/3 to 1) supports between 0 and 1
tree_rescaled <- dendrapply(tree,
 function(x) {
  if (!is.leaf(x))
   attr(x, "probability") <- (3*attr(x, "p") - 1)/2
  x
 })
d5 <- Cophenetic(tree_rescaled, "probability")
cor(d4, d5)

# example of specifying multiple distance measures
d6 <- Cophenetic(tree, c("length", "probability"))
cor(d4, d6)
```

CorrectFrameshifts *Corrects Frameshift Errors In Protein Coding Sequences*

Description

Corrects the reading frame to mitigate the impact of frameshift errors caused by insertions or deletions in unaligned nucleotide sequences.

Usage

```
CorrectFrameshifts(myXStringSet,
                   myAAStringSet,
                   type = "indels",
                   acceptDistance = 0.01,
                   rejectDistance = 0.60,
                   maxComparisons = 10,
                   gapOpening = -13,
                   gapExtension = -1,
                   frameShift = -15,
                   geneticCode = GENETIC_CODE,
                   substitutionMatrix = "PFASUM50",
                   verbose = TRUE,
                   processors = 1)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet of unaligned protein coding sequences in 5' to 3' orientation.
myAAStringSet	An AAStringSet of reference protein sequences. Ideally this would consist of a small set of diverse amino acid sequences belonging to the same group of protein coding sequences as myXStringSet.
type	Character string indicating the type of result desired. This should be one of "indels", "sequences", or "both". (See details section below.)
acceptDistance	Numeric giving the maximum distance from a reference sequence that is acceptable to skip any remaining comparisons.
rejectDistance	Numeric giving the maximum distance from a reference sequence that is allowed when correcting frameshifts. Sequences in myXStringSet that are greater than rejectDistance from the nearest reference sequence will only have their length trimmed from the 3'-end to a multiple of three nucleotides without any frameshift correction.
maxComparisons	The number of reference comparisons to make before stopping the search for a closer reference sequence.
gapOpening	Numeric giving the cost for opening a gap between the query and reference sequences.
gapExtension	Numeric giving the cost for extending an open gap between the query and reference sequences.
frameShift	Numeric giving the cost for shifting between frames of the query sequence.
geneticCode	Named character vector in the same format as GENETIC_CODE (the default), which represents the standard genetic code.
substitutionMatrix	Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment.
verbose	Logical indicating whether to display progress.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.

Details

Accurate translation of protein coding sequences can be greatly disrupted by one or two nucleotide phase shifts that occasionally occur during DNA sequencing. These frameshift errors can potentially be corrected through comparison with other unshifted protein sequences. This function uses a set of reference amino acid sequences (AAStringSet) to find and correct frameshift errors in a set of nucleotide sequences (myXStringSet). First, three frame translation of the nucleotide sequences is performed, and the nearest reference sequence is selected. Then the optimal reading frame at each position is determined based on a variation of the Guan & Uberbacher (1996) method. Putative insertions and/or deletions (indels) are returned in the result, typically with close proximity to the true indel locations. For a comparison of this method to others, see Wang et al. (2013).

If type is "sequences" or "both", then frameshifts are corrected by adding N's and/or removing nucleotides. Note that this changes the nucleotide sequence, and the new sequence often has minor errors because the exact location of the indel(s) cannot be determined. However, the original frameshifts that disrupted the entire downstream sequence are reduced to local perturbations. All of the returned nucleotide sequences will have a reading frame starting from the first position. This allows direct translation, and in practice works well if there is a similar reference myAAStringSet with the correct reading frame. Hence it is more important that myAAStringSet contain a wide diversity of sequences than it is that it contain many sequences.

Multiple inputs control the time required for frameshift correction. The number of sequences in the reference set (myAAStringSet) will affect the speed of the first search for similar sequences. Assessing frameshifts in the second step requires order N*M time, where N and M are the lengths of the query (myXStringSet) and reference sequences. Two parameters control the number of assessments that are made for each sequence: (1) maxComparisons determines the maximum number of reference sequences to compare to each query sequence, and (2) acceptDist defines the maximum distance between a query and reference that is acceptable before continuing to the next query sequence. A lower value for maxComparisons or a higher value for acceptDist will accelerate frameshift correction, potentially at the expense of some accuracy.

Value

If type is "indels" then the returned object is a list with the same length as myXStringSet. Each element is a list with four components:

"insertions"	Approximate positions of inserted nucleotides, which could be removed to correct the reading frame, or excess nucleotides at the 3'-end that make the length longer than a multiple of three.
"deletions"	Approximate positions of deleted nucleotides, which could be added back to correct the reading frame.
"distance"	The amino acid distance from the nearest reference sequence, between 0 and 1.
"index"	The integer index of the reference sequence that was used for frame correction, or 0 if no reference sequence was within rejectDistance.

Note that positions in insertions and deletions are sometimes repeated to indicate that the same position needs to be shifted successively more than once to correct the reading frame.

If type is "sequences" then the returned object is an XStringSet of the same type as the input (myXStringSet). Nucleotides are added or deleted as necessary to correct for frameshifts. The returned sequences all have a reading frame starting from position 1, so that they can be translated directly.

If type is "both" then the returned object is a list with two components: one for the "indels" and the other for the "sequences".

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Guan, X., & Uberbacher, E. C. (1996). Alignments of DNA and protein sequences containing frameshift errors. Computer Applications in the Biosciences : CABIOS, **12(1)**, 31-40.

Wang, Q., et al. (2013). Ecological Patterns of nifH Genes in Four Terrestrial Climatic Zones Explored with Targeted Metagenomics Using FrameBot, a New Informatics Tool. mBio, **4(5)**, e00592-13-e00592-13.

See Also

AlignTranslation, OrientNucleotides, PFASUM

Examples

```
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

# introduce artificial indels
n_ins <- 2 # insertions per sequence
shifted <- replaceAt(dna,
 lapply(width(dna),
  sample,
  n_ins),
 sample(DNA_BASES,
  n_ins,
  replace=TRUE))
n_dels <- 1 # deletions per sequence
shifted <- replaceAt(shifted,
 as(lapply(width(shifted),
  function(x) {
   IRanges(sample(x,
     n_dels),
    width=1)
  }), "IRangesList"))

# to make frameshift correction more challenging,
# only supply 20 reference amino acid sequences
s <- sample(length(dna), 20)
x <- CorrectFrameshifts(shifted,
 translate(dna[s]),
 type="both")

# there was a wide range of distances
# to the nearest reference sequence
quantile(unlist(lapply(x[[1]], `[`, "distance")))

# none of the sequences were > rejectDistance
# from the nearest reference sequence
length(which(unlist(lapply(x[[1]], `[`, "index"))==0))

# the number of indels was generally correct
table(unlist(lapply(x[[1]], function(x) {
```

```
 length(x$insertions)}))))/length(shifted)
table(unlist(lapply(x[[1]], function(x) {
 length(x$deletions)}))))/length(shifted)

# align and display the translations
AA <- AlignTranslation(x$sequences,
 readingFrame=1,
 type="AAStringSet")
BrowseSeqs(AA)
```

`CreateChimeras` *Create Artificial Chimeras*

Description

Creates artificial random chimeras from a set of sequences.

Usage

```
CreateChimeras(myDNAStringSet,
               numChimeras = 10,
               numParts = 2,
               minLength = 80,
               maxLength = Inf,
               minChimericRegionLength = 30,
               randomLengths = TRUE,
               includeParents = TRUE,
               processors = 1,
               verbose = TRUE)
```

Arguments

`myDNAStringSet`	A `DNAStringSet` object with aligned sequences.
`numChimeras`	Number of chimeras desired.
`numParts`	Number of chimeric parts from which to form a single chimeric sequence.
`minLength`	Minimum length of the complete chimeric sequence.
`maxLength`	Maximum length of the complete chimeric sequence.
`minChimericRegionLength`	Minimum length of the chimeric region of each sequence part.
`randomLengths`	Logical specifying whether to create random length chimeras in addition to random breakpoints.
`includeParents`	Whether to include the parents of each chimera in the output.
`processors`	The number of processors to use, or `NULL` to automatically detect and use all available processors.
`verbose`	Logical indicating whether to display progress.

Details

Forms a set of random chimeras from the input set of (typically good quality) sequences. The chimeras are created by merging random sequences at random breakpoints. These chimeras can be used for testing the accuracy of the FindChimeras or other chimera finding functions.

Value

A DNAStringSet object containing chimeras. The names of the chimeras are specified as "parent #1 name [chimeric region] (distance from parent to chimera), ...".

If includeParents = TRUE then the parents of the chimeras are included at the end of the result. The parents are trimmed to the same length as the chimera if randomLengths = TRUE. The names of the parents are specified as "parent #1 name [region] (distance to parent #2, ...)".

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

FindChimeras, Seqs2DB

Run vignette("FindChimeras", package = "DECIPHER") to see a related vignette.

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
chims <- CreateChimeras(dna)
BrowseSeqs(chims)
```

DateTree *Time Calibrate a Dendrogram*

Description

Uses the relative rate framework to time calibrate a phylogenetic tree with known tip dates or root node age.

Usage

```
DateTree(x,
         time = 1,
         penalty = 0,
         epsilon = 1e-06)
```

Arguments

x	An object of class dendrogram representing a rooted and strictly bifurcating phylogenetic tree with branch lengths reflecting evolutionary distance.
time	Either a single number representing the root age, where all the tips are at assumed to be at the same time (i.e., an isochronous tree), or a named vector containing sampling dates for two or more leaves, where the root age is inferred (i.e., a heterochronous tree).
penalty	Either zero (the default) to infer an unconstrained relative rate at each internal node (as in the original algorithm) or a positive regularization penalty. (See details section below.)
epsilon	Numeric giving the minimum assumed branch length, since branch lengths must be non-zero in the relative rate framework.

Details

Temporally resolved phylogenies offer significant advantages for some analyses, because time trees provide date estimates for evolutionary events that can be linked to external knowledge. There are two main types of trees supported by DateTree: (1) the root is ancient relative to the tips, so all tips (i.e., leaves) are presumed to occur at the same time; or (2) the root date is unknown, but the sampling dates of two or more tips are known.

The relative rate framework (Miura *et al.*, 2020) is used to date the input phylogenetic tree (x) given the known time of the root (i.e., node dating) or leaves (i.e., tip dating). If one value is provided for time, then the root of the output dendrogram will be at zero height and all the leaves are assumed to occur at time. If multiple (named) time values are provided, then the corresponding leaves are at these heights and the root height is inferred with linear regression.

A single time can be provided when all leaves are sampled at effectively the same time relative to the (presumably much older) root. This is useful for dating phylogenetic trees over long evolutionary timescales, where only the root age is known and all tips are assumed to be at approximately the same age. The default value (i.e., time = 1) will calibrate the tree into units of relative time (i.e., unit time from zero to one).

Alternatively, time can contain multiple (named) values providing the known sampling dates for leaf labels. The root time is then inferred by linear regression between root-to-tip distance and the known sampling time, which requires that the samples differ across the time scale. This regression is critical for accurate time calibration towards the root, and a strong phylogenetic signal is desirable ($R^2 > 0.3$). The resulting slope represents the rate of evolutionary divergence under a strictly clock-like model.

A relaxed molecular clock is used to make the output tree ultrametric, such that branches represent chronological time rather than evolutionary distance. The method works by projecting internal node distances from the tips toward the root (Miura *et al.*, 2020), which is based on the assumption that the relative rates of branches must be equal where they intersect. In this way, a rate parameter is estimated for each internal node of the tree. Unlike many alternative methods, the relative rate framework does not make prior assumptions about how rates are distributed.

Since the number of inferred rate parameters is high, a regularization penalty can be introduced to encourage more clock-like rates. Under this alternative model, deviations from a strict molecular clock ($rate = 1$) are penalized such that $\text{rate}_{\text{new}} = \text{rate}_{\text{original}}^{1/(1+\text{penalty})}$. Thus, the rate is unaltered when penalty is zero (the default), corresponding to the unconstrained local rate variation in the original

model. Lower penalties place more trust in the branch lengths, whereas higher penalties encourage more clock-like rates. Results on simulated benchmarks suggest that a `penalty` value of `1` generally yields slightly more accurate results than the default value of zero.

Since the underlying algorithm is relative fast, confidence intervals can be established by repeated sampling of the branch lengths and `time` values from appropriate distributions. This approach requires additional information about the expected variance in branch lengths and `time`, as well as any variability in tree topology. It is possible to combine time calibration with bootstrap or jackknife subsampling during tree inference to obtain joint error estimates of node ages. See the examples section below for a demonstration of using variability in sampling `time` to quantify uncertainty in root age.

Several considerations are important for obtaining accurate time trees (Rieux & Balloux, 2016). First, past demographic changes, recombination, or natural selection can interfere with date estimates. Second, root location is central to time projection, since the root polarizes the direction of evolutionary change. When possible, outgroup rooting is preferred, as root placement cannot be disentangled from varying rates of evolution. Third, branch lengths of the input dendrogram (`x`) are expected to be in units proportional to evolutionary change (e.g., substitutions per site). Branch length or topological inaccuracies in the input dendrogram (`x`) will hinder accurate time calibration.

Value

A time calibrated dendrogram (also known as a timetree or chronogram) where the root is at a lower height (further in the past) than the tips. This directionality is reversed from the input dendrogram (`x`), because distance on the output tree represents chronological time that elapsed since the root rather than accumulated evolutionary distance from the tips.

Note

The root of dated trees is always older (lower in value) than the leaves. This time axis is flipped from that of conventional dendrograms, and can be offset from zero depending on the values of `time`. Therefore, it is necessary to specify axis limits whenever plotting dated dendrograms.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Miura S., *et al.* (2020) A new method for inferring timetrees from temporally sampled molecular sequences. *PLOS Comp. Biol.*, **16(1)**, e1007046.

Rieux A. and Balloux F. (2016) Inferences from tip-calibrated phylogenies: a review and a practical guide. *Molecular Ecology*, **25**, 1911-24.

See Also

`Zipline`, `Treeline`, `ReadDendrogram`, `WriteDendrogram`

Examples

```
# read a Newick formatted tree of SARS-CoV-2 relationships
fpath <- system.file("extdata", "SARS-CoV-2.newick", package="DECIPHER")
tree <- ReadDendrogram(fpath)
plot(tree)

# parse sampling dates from labels
times <- labels(tree)
times <- setNames(sapply(strsplit(times, " "), tail, n=1),
 times)
times <- as.Date(times)
times <- times - as.Date("2020-01-01")
times <- setNames(as.numeric(times), names(times))
times <- 2020 + times/365
head(times) # sampling year

# evaluate phylogenetic strength
dists <- attr(tree, "height") - rapply(tree, attr, which="height")
plot(times,
 dists,
 xlab="Sampling time",
 ylab="Root-to-tip distance")
fit <- lm(dists~times)
abline(fit, h=0)
summary(fit)$adj.r.squared # R^2 > 0.3 is preferable

# convert from distances to times
timetree <- DateTree(tree, times)

# need to adjust y-axis limits for plotting
lim <- range(attr(timetree, "height"),
 rapply(timetree, attr, which="height"))
plot(timetree, ylim=rev(lim))

# display sampling time as leaf label
relabeled <- dendrapply(timetree,
 function(x) {
  if (is.leaf(x))
   attr(x, "label") <- format(times[attr(x, "label")])
  x
 })
plot(relabeled, ylim=rev(lim))

# root height represents origin time of SARS-CoV-2
root <- attr(timetree, "height") # mid-November 2019

# measure variability induced by uncertainty in sampling time
est <- numeric(100) # replicate measurements
for (i in seq_along(est)) {
 delta <- sample(c(-1, 0, 1), length(times), replace=TRUE)
 alt_times <- times + delta/365 # shift by +/- 1 day
 est[i] <- attr(DateTree(tree, time=alt_times), "height")
```

```
}
hist(est, xlab="Estimated uncertainty in root time")
abline(v=root, lwd=2, lty=2) # unperturbed estimate

# timetree for samples sharing an ancient ancestor
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
x <- Treeline(dna, model="TN93+F")
plot(x, leaflab="none") # not ultrametric
y <- DateTree(x)
plot(y, leaflab="none", ylim=c(1, 0))
```

DB2Seqs *Export Database Sequences to a FASTA or FASTQ File*

Description

Exports a database containing sequences to a FASTA or FASTQ formatted file of sequence records.

Usage

```
DB2Seqs(file,
        dbFile,
        tblName = "Seqs",
        identifier = "",
        type = "BStringSet",
        limit = -1,
        replaceChar = NA,
        nameBy = "description",
        orderBy = "row_names",
        removeGaps = "none",
        append = FALSE,
        width = 80,
        compress = FALSE,
        chunkSize = 1e5,
        sep = "::",
        clause = "",
        processors = 1,
        verbose = TRUE)
```

Arguments

file	Character string giving the location where the file should be written.
dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table in which to extract the data.

identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected.
type	The type of XStringSet (sequences) to export to a FASTA formatted file or QualityScaledXStringSet to export to a FASTQ formatted file. This should be (an unambiguous abbreviation of) one of "DNAStringSet", "RNAStringSet", "AAStringSet", "BStringSet", "QualityScaledDNAStringSet", "QualityScaledRNAStringSet", "QualityScaledAAStringSet", or "QualityScaledBStringSet". (See details section below.)
limit	Number of results to display. The default (-1) does not limit the number of results.
replaceChar	Optional character used to replace any characters of the sequence that are not present in the XStringSet's alphabet. Not applicable if type=="BStringSet". The default (NA) results in an error if an incompatible character exist. (See details section below.)
nameBy	Character string giving the column name(s) for identifying each sequence record. If more than one column name is provided, the information in each column is concatenated, separated by sep, in the order specified.
orderBy	Character string giving the column name for sorting the results. Defaults to the order of entries in the database. Optionally can be followed by " ASC" or " DESC" to specify ascending (the default) or descending order.
removeGaps	Determines how gaps ("-" or "." characters) are removed in the sequences. This should be (an unambiguous abbreviation of) one of "none", "all" or "common".
append	Logical indicating whether to append the output to the existing file.
width	Integer specifying the maximum number of characters per line of sequence. Not applicable when exporting to a FASTQ formatted file.
compress	Logical specifying whether to compress the output file using gzip compression.
chunkSize	Number of sequences to write to the file at a time. Cannot be less than the total number of sequences if removeGaps is "common".
sep	Character string providing the separator between fields in each sequence's name, by default pairs of colons ("::").
clause	An optional character string to append to the query as part of a "where clause".
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display status.

Details

Sequences are exported into either a FASTA or FASTQ file as determined by the type of sequences. If type is an XStringSet then sequences are exported to FASTA format. Quality information for QualityScaledXStringSets are interpreted as PredQuality scores before export to FASTQ format.

If type is "BStringSet" (the default) then sequences are exported to a FASTA file exactly the same as they were when imported. If type is "DNAStringSet" then all U's are converted to T's before export, and vice versa if type is "RNAStringSet". All remaining characters not in the XStringSet's alphabet are converted to replaceChar or removed if replaceChar is "". Note that if replaceChar is NA (the default), it will result in an error when an unexpected character is found.

Value

Writes a FASTA or FASTQ formatted file containing the sequence records in the database.

Returns the number of sequence records written to the `file`.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 tf <- tempfile()
 DB2Seqs(tf, db, limit=10)
 file.show(tf) # press 'q' to exit
 unlink(tf)
}
```

---

deltaGrules — *Free Energy of Hybridization of Probe/Target Quadruplets on Microarrays*

Description

An 8D array with four adjacent base pairs of the probe and target sequences at a time. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "T", or "-"). The array contains the standard Gibbs free energy change of probe binding (dG, [kcal/mol]) for every quadruple base pairing.

Usage

```
data(deltaGrules)
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -0.141 0 0 0 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ...

Details

The first four dimensions correspond to the four probe positions from 5' to 3'. The fifth to eighth dimensions correspond to the four positions from 5' to 3' of the target sequence.

Source

Data obtained using NimbleGen microarrays and a Linear Free Energy Model developed by Yilmaz *et al*.

References

Yilmaz LS, Loy A, Wright ES, Wagner M, Noguera DR (2012) Modeling Formamide Denaturation of Probe-Target Hybrids for Improved Microarray Probe Design in Microbial Diagnostics. PLoS ONE 7(8): e43862. doi:10.1371/journal.pone.0043862.

Examples

```
data(deltaGrules)
# dG of probe = AGCT / target = A-CT pairing
deltaGrules["A", "G", "C", "T", "A", "-", "C", "T"]
```

---

deltaGrulesRNA *Pseudoenergy Parameters for RNA Quadruplets*

Description

An 8D array with four adjacent base pairs of the RNA duplex. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "U", or "-"). The array contains the pseudoenergy of duplex formation for every quadruple base pairing.

Usage

```
data("deltaGrulesRNA")
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -0.776 -0.197 -0.197 -0.291 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ...

Details

The first four dimensions correspond to the four top strand positions from 5' to 3'. The fifth to eighth dimensions correspond to the four bottom strand positions from 5' to 3'.

Source

Psuedoenergy values of ungapped quadruplets are inferred from their log-odds of being found in palindromes within hairpin regions across thousands of non-coding RNA alignments. Each value represents the log-odds of *in vivo* folding relative to chance.

Examples

```
data(deltaGrulesRNA)
# dG of the duplex AGCU / ACCU pairing (1 mismatch)
deltaGrulesRNA["A", "G", "C", "U", "A", "C", "C", "U"]
```

`deltaHrules` *Change in Enthalpy of Hybridization of DNA/DNA Quadruplets in Solution*

Description

An 8D array with four adjacent base pairs of the DNA duplex. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "T", or "-"). The array contains the standard enthalpy change of probe binding (dH, [kcal/mol]) for every quadruple base pairing.

Usage

```
data(deltaHrules)
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -7.97 0 0 0 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ...

Details

The first four dimensions correspond to the four top strand positions from 5' to 3'. The fifth to eighth dimensions correspond to the four bottom strand positions from 5' to 3'.

Source

Data from a variety of publications by SantaLucia *et al*.

References

SantaLucia, J., Jr., & Hicks, D. (2004) The Thermodynamics of DNA Structural Motifs. Annual Review of Biophysics and Biomolecular Structure, 33(1), 415-440. doi:10.1146/annurev.biophys.32.110601.141800.

Examples

```
data(deltaHrules)
# dH of the duplex AGCT / A-CT pairing
deltaHrules["A", "G", "C", "T", "A", "-", "C", "T"]
```

deltaHrulesRNA *Change in Enthalpy of Hybridization of RNA/RNA Quadruplets in Solution*

Description

An 8D array with four adjacent base pairs of the RNA duplex. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "U", or "-"). The array contains the standard enthalpy change of probe binding (dH, [kcal/mol]) for every quadruple base pairing.

Usage

```
data(deltaHrulesRNA)
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -6.55 0 0 0 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ...

Details

The first four dimensions correspond to the four top strand positions from 5' to 3'. The fifth to eighth dimensions correspond to the four bottom strand positions from 5' to 3'.

Source

Data from a variety of publications by SantaLucia *et al*.

References

SantaLucia, J., Jr., & Hicks, D. (2004) The Thermodynamics of DNA Structural Motifs. Annual Review of Biophysics and Biomolecular Structure, 33(1), 415-440. doi:10.1146/annurev.biophys.32.110601.141800.

Examples

```
data(deltaHrulesRNA)
# dH of the duplex AGCU / A-CU pairing
deltaHrulesRNA["A", "G", "C", "U", "A", "-", "C", "U"]
```

`deltaSrules` *Change in Entropy of Hybridization of DNA/DNA Quadruplets in Solution*

Description

An 8D array with four adjacent base pairs of the DNA duplex. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "T", or "-"). The array contains the standard entropy change of probe binding (dS, [kcal/mol]) for every quadruple base pairing.

Usage

```
data(deltaSrules)
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -0.0226 0 0 0 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ... ..$: chr [1:5] "A" "C" "G" "T" ...

Details

The first four dimensions correspond to the four top strand positions from 5' to 3'. The fifth to eighth dimensions correspond to the four bottom strand positions from 5' to 3'.

Source

Data from a variety of publications by SantaLucia *et al*.

References

SantaLucia, J., Jr., & Hicks, D. (2004) The Thermodynamics of DNA Structural Motifs. Annual Review of Biophysics and Biomolecular Structure, 33(1), 415-440. doi:10.1146/annurev.biophys.32.110601.141800.

Examples

```
data(deltaSrules)
# dS of the duplex AGCT / A-CT pairing
deltaSrules["A", "G", "C", "T", "A", "-", "C", "T"]
```

deltaSrulesRNA *Change in Entropy of Hybridization of RNA/RNA Quadruplets in Solution*

Description

An 8D array with four adjacent base pairs of the RNA duplex. Each dimension has five elements defining the nucleotide at that position ("A", "C", "G", "T", or "-"). The array contains the standard entropy change of probe binding (dS, [kcal/mol]) for every quadruple base pairing.

Usage

```
data(deltaSrulesRNA)
```

Format

The format is: num [1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5] -0.0182 0 0 0 0 ... - attr(*, "dimnames")=List of 8 ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ... ..$: chr [1:5] "A" "C" "G" "U" ...

Details

The first four dimensions correspond to the four top strand positions from 5' to 3'. The fifth to eighth dimensions correspond to the four bottom strand positions from 5' to 3'.

Source

Data from a variety of publications by SantaLucia *et al*.

References

SantaLucia, J., Jr., & Hicks, D. (2004) The Thermodynamics of DNA Structural Motifs. Annual Review of Biophysics and Biomolecular Structure, 33(1), 415-440. doi:10.1146/annurev.biophys.32.110601.141800.

Examples

```
data(deltaSrulesRNA)
# dS of the duplex AGCU / A-CU pairing
deltaSrulesRNA["A", "G", "C", "U", "A", "-", "C", "U"]
```

DesignArray *Design a Set of DNA Microarray Probes for Detecting Sequences*

Description

Chooses the set of microarray probes maximizing sensitivity and specificity to each target consensus sequence.

Usage

```
DesignArray(myDNAStringSet,
            maxProbeLength = 24,
            minProbeLength = 20,
            maxPermutations = 4,
            numRecordedMismatches = 500,
            numProbes = 10,
            start = 1,
            end = NULL,
            maxOverlap = 5,
            hybridizationFormamide = 10,
            minMeltingFormamide = 15,
            maxMeltingFormamide = 20,
            minScore = -1e+12,
            processors = 1,
            verbose = TRUE)
```

Arguments

myDNAStringSet	A DNAStringSet object of aligned consensus sequences.
maxProbeLength	The maximum length of probes, not including the poly-T spacer. Ideally less than 27 nucleotides.
minProbeLength	The minimum length of probes, not including the poly-T spacer. Ideally more than 18 nucleotides.
maxPermutations	The maximum number of probe permutations required to represent a target site. For example, if a target site has an 'N' then 4 probes are required because probes cannot be ambiguous. Typically fewer permutations are preferably because this requires less space on the microarray and simplifies interpretation of the results.
numRecordedMismatches	The maximum number of recorded potential cross-hybridizations for any target site.
numProbes	The target number of probes on the microarray per input consensus sequence.
start	Integer specifying the starting position in the alignment where potential forward primer target sites begin. Preferably a position that is included in most sequences in the alignment.

end	Integer specifying the ending position in the alignment where potential reverse primer target sites end. Preferably a position that is included in most sequences in the alignment.
maxOverlap	Maximum overlap in nucleotides between target sites on the sequence.
hybridizationFormamide	The formamide concentration (%, vol/vol) used in hybridization at 42 degrees Celsius. Note that this concentration is used to approximate hybridization efficiency of cross-amplifications.
minMeltingFormamide	The minimum melting point formamide concentration (%, vol/vol) of the designed probes. The melting point is defined as the concentration where half of the template is bound to probe.
maxMeltingFormamide	The maximum melting point formamide concentration (%, vol/vol) of the designed probes. Must be greater than the minMeltingFormamide.
minScore	The minimum score of designed probes before exclusion. A greater minScore will accelerate the code because more target sites will be excluded from consideration. However, if the minScore is too high it will prevent any target sites from being recorded.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

The algorithm begins by determining the optimal length of probes required to meet the input constraints while maximizing sensitivity to the target consensus sequence at the specified hybridization formamide concentration. This set of potential target sites is then scored based on the possibility of cross-hybridizing to the other non-target sequences. The set of probes is returned with the minimum possibility of cross-hybridizing.

Value

A data.frame with the optimal set of probes matching the specified constraints. Each row lists the probe's target sequence (name), start position, length in nucleotides, start and end position in the sequence alignment, number of permutations, score, melt point in percent formamide at 42 degrees Celsius, hybridization efficiency (hyb_eff), target site, and probe(s). Probes are designed such that the stringency is determined by the equilibrium hybridization conditions and not subsequent washing steps.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2013) Identification of Bacterial and Archaeal Communities From Source to Tap. Water Research Foundation, Denver, CO.

DR Noguera, et al. (2014). Mathematical tools to optimize the design of oligonucleotide probes and primers. Applied Microbiology and Biotechnology. doi:10.1007/s00253-014-6165-x.

See Also

Array2Matrix, NNLS

Run `vignette("DesignMicroarray", package = "DECIPHER")` to see a related vignette.

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
names(dna) <- 1:length(dna)
probes <- DesignArray(dna)
probes[1,]
```

DesignPrimers *Design PCR Primers Targeting a Specific Group of Sequences*

Description

Assists in the design of primer sets targeting a specific group of sequences while minimizing the potential to cross-amplify other groups of sequences.

Usage

```
DesignPrimers(tiles,
              identifier = "",
              start = 1,
              end = NULL,
              minLength = 17,
              maxLength = 26,
              maxPermutations = 4,
              minCoverage = 0.9,
              minGroupCoverage = 0.2,
              annealingTemp = 64,
              P = 4e-07,
              monovalent = 0.07,
              divalent = 0.003,
              dNTPs = 8e-04,
              minEfficiency = 0.8,
              worstScore = -Inf,
              numPrimerSets = 0,
              minProductSize = 75,
              maxProductSize = 1200,
              maxSearchSize = 1500,
              batchSize = 1000,
```

```
                maxDistance = 0.4,
                primerDimer = 1e-07,
                ragged5Prime = TRUE,
                taqEfficiency = TRUE,
                induceMismatch = FALSE,
                processors = 1,
                verbose = TRUE)
```

Arguments

tiles	A set of tiles representing each group of sequences, as in the format created by the function TileSeqs.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. Determines the target group(s) for which primers will be designed. If "" then all identifiers are selected.
start	Integer specifying the starting position in the alignment where potential forward primer target sites begin. Preferably a position that is included in most sequences in the alignment.
end	Integer specifying the ending position in the alignment where potential reverse primer target sites end. Preferably a position that is included in most sequences in the alignment.
minLength	Integer providing the minimum length of primers to consider in the design.
maxLength	Integer providing the maximum length of primers to consider in the design, which must be less than or equal to the maxLength of tiles.
maxPermutations	Integer providing the maximum number of allowable forward or reverse primer permutations.
minCoverage	Numeric giving the minimum fraction of the target group's sequences that must be covered with the primer set.
minGroupCoverage	Numeric giving the minimum fraction of the target group that must have sequence information (not terminal gaps) in the region covered by the primer set.
annealingTemp	Numeric indicating the desired annealing temperature that will be used in the PCR experiment.
P	Numeric giving the molar concentration of primers in the reaction.
monovalent	The molar concentration of monovalent ([Na] and [K]) ions in solution that will be used to determine a sodium equivalent concentration.
divalent	The molar concentration of divalent ([Mg]) ions in solution that will be used to determine a sodium equivalent concentration.
dNTPs	Numeric giving the molar concentration of free nucleotides added to the solution that will be used to determine a sodium equivalent concentration.
minEfficiency	Numeric giving the minimum efficiency of hybridization desired for the primer set. Note that an efficiency of 99% (0.99) will greatly lower predicted specificity of the primer set, however an efficiency of 50% (0.5) may be too low in actuality to amplify the target group due to error in melt temperature predictions.

worstScore	Numeric specifying the score cutoff to remove target sites from consideration. For example, a worstScore of -5 will remove all primer sets scoring below -5, although this may eventually result in no primer sets meeting the design criteria.
numPrimerSets	Integer giving the optimal number of primer sets (forward and reverse primer sets) to design. If set to zero then all possible forward and reverse primers are returned, but the primer sets minimizing potential cross-amplifications are not chosen.
minProductSize	Integer giving the minimum number of nucleotides desired in the PCR product.
maxProductSize	Integer giving the maximum number of nucleotides desired in the PCR product.
maxSearchSize	Integer giving the maximum number of nucleotides to search for false priming upstream and downstream of the expected binding site.
batchSize	Integer specifying the number of primers to simulate hybridization per batch that is passed to CalculateEfficiencyPCR.
maxDistance	Numeric specifying the maximal fraction of mismatched base pairings on a rolling basis beginning from the 3' end of the primer.
primerDimer	Numeric giving the maximum amplification efficiency of potential primer-dimer products.
ragged5Prime	Logical specifying whether the 5' end or 3' end of primer permutations targeting the same site should be varying lengths.
taqEfficiency	Logical determining whether to make use of elongation efficiency and maxDistance to increase predictive accuracy for *Taq* DNA Polymerase amplifying primers with mismatches near the 3' terminus. Note that this should be set to FALSE if using a high-fidelity polymerase with 3' to 5' exonuclease activity.
induceMismatch	Logical or integer specifying whether to induce a mismatch in the primer with the template DNA. If TRUE then a mismatch is induced at the 6th primer position. If an integer value is provided between 2 and 6 then a mismatch is induced in that primer position, where the 3'-end is defined as position 1.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

Primers are designed for use with *Taq* DNA Polymerase to maximize sensitivity and specificity for the target group of sequences. The design makes use of *Taq*'s bias against certain 3' terminal mismatch types in order to increase specificity further than can be achieve with hybridization efficiency alone.

Primers are designed from a set of tiles to target each identifier while minimizing affinity for all other tiled groups. Arguments provide constraints that ensure the designed primer sets meet the specified criteria as well as being optimized for the particular experimental conditions. A search is conducted through all tiles in the same alignment position to estimate the chance of cross-amplification with a non-target group.

If numPrimers is greater than or equal to one then the set of forward and reverse primers that minimizes potential false positive overlap is returned. This will also initiate a thorough search

through all target sites upstream and downstream of the expected binding sites to ensure that the primers do not bind to nearby positions. Lowering the `maxSearchSize` will speed up the thorough search at the expense of potentially missing an unexpected target site. The number of possible primer sets assessed is increased with the size of `numPrimers`.

Value

A different `data.frame` will be returned depending on number of primer sets requested. If no primer sets are required then columns contain the forward and reverse primers for every possible position scored by their potential to amplify other identified groups. If one or more primer sets are requested then columns contain information for the optimal set of forward and reverse primers that could be used in combination to give the fewest potential cross-amplifications.

Note

The program OligoArrayAux (http://www.unafold.org/Dinamelt/software/oligoarrayaux.php) must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the R console:

```
system("hybrid-min -V")
```

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2013) "Exploiting Extension Bias in PCR to Improve Primer Specificity in Ensembles of Nearly Identical DNA Templates." Environmental Microbiology, doi:10.1111/1462-2920.12259.

See Also

AmplifyDNA, CalculateEfficiencyPCR, DesignSignatures, TileSeqs

Run `vignette("DesignPrimers", package = "DECIPHER")` to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 # not run (must have OligoArrayAux installed first):
 ## Not run: tiles <- TileSeqs(db,
  identifier=c("Rhizobiales", "Sphingomonadales"))
## End(Not run)
 ## Not run: primers <- DesignPrimers(tiles, identifier="Rhizobiales",
  start=280, end=420, minProductSize=50, numPrimerSets=1)
## End(Not run)
}
```

DesignProbes *Design FISH Probes Targeting a Specific Group of Sequences*

Description

Assists in the design of single or dual probes targeting a specific group of sequences while minimizing the potential to cross-hybridize with other groups of sequences.

Usage

```
DesignProbes(tiles,
             identifier = "",
             start = 1,
             end = NULL,
             minLength = 17,
             maxLength = 26,
             maxPermutations = 4,
             minCoverage = 0.9,
             minGroupCoverage = 0.2,
             hybTemp = 46,
             P = 2.5e-07,
             Na = 1,
             FA = 35,
             minEfficiency = 0.5,
             worstScore = -Inf,
             numProbeSets = 0,
             batchSize = 1000,
             target = "SSU",
             verbose = TRUE)
```

Arguments

tiles	A set of tiles representing each group of sequences, as in the format created by the function TileSeqs.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. Determines the target group(s) for which probes will be designed. If "" then all identifiers are selected.
start	Integer specifying the starting position in the alignment where potential target sites begin. Preferably a position that is included in most sequences in the alignment.
end	Integer specifying the ending position in the alignment where potential target sites end. Preferably a position that is included in most sequences in the alignment.
minLength	Integer providing the minimum length of probes to consider in the design.
maxLength	Integer providing the maximum length of probes to consider in the design, which must be less than or equal to the maxLength of tiles.

`maxPermutations`
: Integer providing the maximum number of probe permutations required to reach the desired coverage of a target site.

`minCoverage`
: Numeric giving the minimum fraction of the target group's sequences that must be covered by the designed probe(s).

`minGroupCoverage`
: Numeric giving the minimum fraction of the target group that must have sequence information (not terminal gaps) in the target site's region.

`hybTemp`
: Numeric specifying the hybridization temperature, typically `46` degrees Celsius.

`P`
: Numeric giving the molar concentration of probes during hybridization.

`Na`
: Numeric giving the molar sodium concentration in the hybridization buffer. Values may range between 0.01M and 1M. Note that salt correction from 1 molar is not available for the thermodynamic rules of RNA/RNA interactions.

`FA`
: Numeric concentration (as percent v/v) of the denaturant formamide in the hybridization buffer.

`minEfficiency`
: Numeric giving the minimum equilibrium hybridization efficiency desired for designed probe(s) at the defined experimental conditions.

`worstScore`
: Numeric specifying the score cutoff to remove target sites from consideration. For example, a `worstScore` of -5 will remove all probes scoring below -5, although this may eventually result in no probes meeting the design criteria.

`numProbeSets`
: Integer giving the optimal number of dual probe sets to design. If set to zero then all potential single probes are returned, and the probe sets minimizing potential false cross-hybridizations are not chosen.

`batchSize`
: Integer specifying the number of probes to simulate hybridization per batch that is passed to `CalculateEfficiencyFISH`.

`target`
: The target molecule used in the generation of tiles. Either `"SSU"` for the small-subunit rRNA, `"LSU"` for the large-subunit rRNA, or "Other". Used to determine the domain for `dG3` calculations, which is plus or minus 200 nucleotides of the target site if `"Other"`.

`verbose`
: Logical indicating whether to display progress.

Details

Probes are designed to maximize sensitivity and specificity to the target group(s) (`identifier`(s)). If `numProbeSets` > `0` then that many pairs of probes with minimal cross-hybridization overlap are returned, enabling increased specificity with a dual-color approach.

Probes are designed from a set of tiles to target each `identifier` while minimizing affinity for all other tiled groups. Arguments provide constraints that ensure the designed probes meet the specified criteria as well as being optimized for the particular experimental conditions. A search is conducted through all tiles in the same alignment position to estimate the chance of cross-hybridization with a non-target group.

Two models are used in design, both of which were experimentally calibrated using denaturation profiles from 5 organisms belonging to all three domains of life. Probe lengths are chosen to meet the `minEfficiency` using a fast model of probe-target hybridization. Candidate probes are then

confirmed using a slower model that also takes into account probe-folding and target-folding. Finally, probes are scored for their inability to cross-hybridize with non-target groups by using the fast model and taking into account any mismatches.

Value

A different `data.frame` will be returned depending on number of primer sets requested. If no probe sets are required then columns contain the designed probes for every possible position scored by their potential to cross-hybridize with other identified groups. If one or more probe sets are requested then columns contain information for the optimal set of probes (probe one and probe two) that could be used in combination to give the fewest potential cross-hybridizations.

Note

The program OligoArrayAux (http://www.unafold.org/Dinamelt/software/oligoarrayaux.php) must be installed in a location accessible by the system. For example, the following code should print the installed OligoArrayAux version when executed from the R console:

```
system("hybrid-min -V")
```

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2014) "Automated Design of Probes for rRNA-Targeted Fluorescence In Situ Hybridization Reveals the Advantages of Using Dual Probes for Accurate Identification." Applied and Environmental Microbiology, doi:10.1128/AEM.01685-14.

See Also

CalculateEfficiencyFISH, TileSeqs

Run `vignette("DesignProbes", package = "DECIPHER")` to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 # not run (must have OligoArrayAux installed first):
 ## Not run: tiles <- TileSeqs(db,
  identifier=c("Rhizobiales", "Sphingomonadales"))
## End(Not run)
 ## Not run: probes <- DesignProbes(tiles, identifier="Rhizobiales",
  start=280, end=420)
## End(Not run)
}
```

DesignSignatures *Design PCR Primers for Amplifying Group-Specific Signatures*

Description

Aids the design of pairs of primers for amplifying a unique "signature" from each group of sequences. Signatures are distinct PCR products that can be differentiated by their length, melt temperature, or sequence.

Usage

```
DesignSignatures(dbFile,
                 tblName = "Seqs",
                 identifier = "",
                 focusID = NA,
                 type = "melt",
                 resolution = 0.5,
                 levels = 10,
                 enzymes = NULL,
                 minLength = 17,
                 maxLength = 26,
                 maxPermutations = 4,
                 annealingTemp = 64,
                 P = 4e-07,
                 monovalent = 0.07,
                 divalent = 0.003,
                 dNTPs = 8e-04,
                 minEfficiency = 0.8,
                 ampEfficiency = 0.5,
                 numPrimerSets = 100,
                 minProductSize = 70,
                 maxProductSize = 400,
                 kmerSize = 8,
                 searchPrimers = 500,
                 maxDictionary = 20000,
                 primerDimer = 1e-07,
                 pNorm = 1,
                 taqEfficiency = TRUE,
                 processors = 1,
                 verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the DNA sequences are located.

identifier	Optional character string used to narrow the search results to those matching a specific identifier. Determines the target group(s) for which primers will be designed. If "" then all identifiers are selected.
focusID	Optional character string specifying which of the identifiers will be used in the initial step of designing primers. If NA (the default), then the identifier with the most sequence information is used as the focusID.
type	Character string indicating the type of signature being used to differentiate the PCR products from each group. This should be one of "melt", "length", or "sequence".
resolution	Numeric specifying the "resolution" of the experiment, or a vector giving the boundaries of bins. (See details section below.)
levels	Numeric giving the number of "levels" that can be distinguished in each bin. (See details section below.)
enzymes	Named character vector providing the cut sites of one or more restriction enzymes. Cut sites must be delineated in the same format as RESTRICTION_ENZYMES.
minLength	Integer providing the minimum length of primers to consider in the design.
maxLength	Integer providing the maximum length of primers to consider in the design.
maxPermutations	Integer providing the maximum number of permutations allowed in a forward or reverse primer to attain greater coverage of sequences.
annealingTemp	Numeric indicating the desired annealing temperature that will be used in the PCR experiment.
P	Numeric giving the molar concentration of primers in the reaction.
monovalent	The molar concentration of monovalent ([Na] and [K]) ions in solution that will be used to determine a sodium equivalent concentration.
divalent	The molar concentration of divalent ([Mg]) ions in solution that will be used to determine a sodium equivalent concentration.
dNTPs	Numeric giving the molar concentration of free nucleotides added to the solution that will be used to determine a sodium equivalent concentration.
minEfficiency	Numeric giving the minimum efficiency of hybridization desired for the primer set.
ampEfficiency	Numeric giving the minimum efficiency required for theoretical amplification of the primers. Note that ampEfficiency must be less than or equal to minEfficiency. Lower values of ampEfficiency will allow for more PCR products, although very low values are unrealistic experimentally.
numPrimerSets	Integer giving the optimal number of primer sets (forward and reverse primer sets) to design.
minProductSize	Integer giving the minimum number of nucleotides desired in the PCR product.
maxProductSize	Integer giving the maximum number of nucleotides desired in the PCR product.
kmerSize	Integer giving the size of k-mers to use in the preliminary search for potential primers.
searchPrimers	Numeric specifying the number of forward and reverse primers to use in searching for potential PCR products. A lower value will result in a faster search, but potentially neglect some useful primers.

maxDictionary	Numeric giving the maximum number of primers to search for simultaneously in any given step.
primerDimer	Numeric giving the maximum amplification efficiency of potential primer-dimer products.
pNorm	Numeric specifying the power (p > 0) used in calculating the L^p-norm when scoring primer pairs. By default (p = 1), the score is equivalent to the average difference between pairwise signatures. When p < 1, many small differences will be preferred over fewer large differences, and vice versa when p > 1. This enables prioritizing primer pairs that will yield a greater number of unique signatures (p < 1), or fewer distinct, but more dissimilar, signatures (p > 1).
taqEfficiency	Logical determining whether to make use of elongation efficiency to increase predictive accuracy for *Taq* DNA Polymerase amplifying primers with mismatches near the 3' terminus. Note that this should be set to FALSE if using a high-fidelity polymerase with 3' to 5' exonuclease activity.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

Signatures are group-specific PCR products that can be differentiated by either their melt temperature profile, length, or sequence. DesignSignatures assists in finding the optimal pair of forward and reverse primers for obtaining a distinguishable signature from each group of sequences. Groups are delineated by their unique identifier in the database. The algorithm works by progressively narrowing the search for optimal primers: (1) the most frequent k-mers are found; (2) these are used to design primers initially matching the focusID group; (3) the most common forward and reverse primers are selected based on all of the groups, and ambiguity is added up to maxPermutations; (4) a final search is performed to find the optimal forward and reverse primer. Pairs of primers are scored by the distance among the signatures generated for each group, which depends on the type of experiment.

The arguments resolution and levels control the theoretical resolving power of the experiment. The signature for a group is discretized or grouped into "bins" each with a certain magnitude of the signal. Here resolution determines the separation between distinguishable "bins", and levels controls the range of values in each bin. A high-accuracy experiment would have many bins and/or many levels. While levels is interpreted similarly for every type of experiment, resolution is treated differently depending on type. If type is "melt", then resolution can be either a vector of different melt temperatures, or a single number giving the change in temperatures that can be differentiated. A high-resolution melt (HRM) assay would typically have a resolution between 0.25 and 1 degree Celsius. If type is "length" then resolution is either the number of bins between the minProductSize and maxProductSize, or the bin boundaries. For example, resolution can be lower (wider bins) at long lengths, and higher (narrower bins) at shorter lengths. If type is "sequence" then resolution sets the k-mer size used in differentiating amplicons. Oftentimes, 4 to 6-mers are used for the classification of amplicons.

The signatures can be diversified by using a restriction enzyme to digest the PCR products when type is "melt" or "length". If enzymes are supplied then the an additional search is made to find the best enzyme to use with each pair of primers. In this case, the output includes all of the primer

pairs, as well as any enzymes that will digest the PCR products of that primer pair. The output is re-scored to rank the top primer pair and enzyme combination. Note that enzymes is inapplicable when type is "sequence" because restriction enzymes do not alter the sequence of the DNA. Also, it is recommended that only a subset of the available RESTRICTION_ENZYMES are used as input enzymes in order to accelerate the search for the best enzyme.

Value

A data.frame with the top-scoring pairs of forward and reverse primers, their score, the total number of PCR products, and associated columns for the restriction enzyme (if enzyme is not NULL).

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E.S. & Vetsigian, K.H. (2016) "DesignSignatures: a tool for designing primers that yields amplicons with distinct signatures." Bioinformatics, doi:10.1093/bioinformatics/btw047.

See Also

AmplifyDNA, CalculateEfficiencyPCR, DesignPrimers, DigestDNA, Disambiguate, MeltDNA, RESTRICTION_ENZYMES

Run vignette("DesignSignatures", package = "DECIPHER") to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 # below are suggested inputs for different types of experiments
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")

 ## Not run:
 # High Resolution Melt (HRM) assay:
 primers <- DesignSignatures(db,
            resolution=seq(75, 100, 0.25), # degrees Celsius
            minProductSize=55, # base pairs
            maxProductSize=400)

 # Primers for next-generation sequencing:
 primers <- DesignSignatures(db,
            type="sequence",
            minProductSize=300, # base pairs
            maxProductSize=700,
            resolution=5, # 5-mers
            levels=5)

 # Primers for community fingerprinting:
 primers <- DesignSignatures(db,
            type="length",
            levels=2, # presence/absence
```

```
            minProductSize=200, # base pairs
            maxProductSize=1400,
            resolution=c(seq(200, 700, 3),
                         seq(705, 1000, 5),
                         seq(1010, 1400, 10)))

 # Primers for restriction fragment length polymorphism (RFLP):
 data(RESTRICTION_ENZYMES)
 myEnzymes <- RESTRICTION_ENZYMES[c("EcoRI", "HinfI", "SalI")]
 primers <- DesignSignatures(db,
            type="length",
            levels=2, # presence/absence
            minProductSize=200, # base pairs
            maxProductSize=600,
            resolution=c(seq(50, 100, 3),
                         seq(105, 200, 5),
                         seq(210, 600, 10)),
            enzymes=myEnzymes)

## End(Not run)
}
```

DetectRepeats *Detect Repeats in a Sequence*

Description

Detects approximate copies of sequence patterns that likely arose from duplication events and therefore share a common ancestor.

Usage

```
DetectRepeats(myXStringSet,
              type = "tandem",
              minScore = 8,
              allScores = FALSE,
              maxCopies = 1000,
              maxPeriod = 2000,
              maxFailures = 3,
              maxShifts = 5,
              alphabet = AA_REDUCED[[152]],
              useEmpirical = TRUE,
              correctBackground = TRUE,
              processors = 1,
              verbose = TRUE,
              ...)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of unaligned sequences.
type	Character string indicating the type of repeats to detect. This should be one of "tandem", "interspersed", or "both". Only "tandem" is possible when myXStringSet is an AAStringSet. (See details section below.)
minScore	Numeric giving the minimum score of repeats in myXStringSet to report.
allScores	Logical specifying whether all repeats should be returned (TRUE) or only the top scoring repeat when there are multiple overlapping matches in the same region.
maxCopies	Numeric defining the maximum copy number of tandem repeat. Since alignment complexity is quadratic in the number of repeat copies, setting a limit on the repeat copy number prevents very long repeats from becoming rate limiting.
maxPeriod	Numeric indicating the maximum periodicity of tandem repeats to consider. Interspersed repeats will only be detected that are at least maxPeriod nucleotides apart. Lower values may increase the sensitivity of tandem repeat detection, and vice versa for higher values.
maxFailures	Numeric determining the maximum number of failing attempts to extend a repeat that are permitted. Numbers greater than zero may increase accuracy at the expense of speed, with decreasing marginal returns as maxFailures gets higher.
maxShifts	Numeric determining the maximum number of failing attempts to shift a repeat left or right that are permitted. Numbers greater than zero may increase accuracy at the expense of speed, with decreasing marginal returns as maxShifts gets higher.
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between sequences. A non-reduced amino acid alphabet can be used by setting alphabet equal to AA_STANDARD. Only applicable if myXStringSet is an AAStringSet.
useEmpirical	Logical specifying whether to use empirical signals of structurally-determined tandem repeats when scoring. Empirical signals include the number of repeats, their periodicity, and their amino acid composition when myXStringSet is an AAStringSet.
correctBackground	Logical controlling whether to correct the substitution matrix for the background distribution of letter frequencies on a per sequence basis.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Further arguments to be passed directly to FindSynteny if type is "interspersed" or "both".

Details

Many sequences are composed of a substantial fraction of repetitive sequence. Two main forms of repetition are tandem repeats and interspersed repeats, which can be caused by duplication events

followed by divergence. Detecting duplications is challenging because of variability in repeat length and composition due to evolution. The significance of a repeat can be quantified by its time since divergence from a common ancestor. DetectRepeats uses a seed-and-extend approach to identify candidate repeats, and tests whether a set of repeats is statistically significant using a background-corrected substitution matrix and gap (i.e., insertion and deletion) penalties. A higher score implies the repeats are more conserved and, therefore, are more likely to have diverged within a finite amount of time from a common ancestor. When myXStringSet is an AAStringSet, similarity includes agreement among predicted secondary structures.

Two possible types of repeats are detectable: * type is "tandem" (the default) or "both" Contiguous approximate copies of a nucleotide or amino acid sequence. First, repeated k-mers are identified along the length of the input sequence(s). Once a k-mer seed is identified, repeated attempts are made to extend the repeat left and right, as well as optimize the beginning and ending positions. * type is "interspersed" or "both" Dispersed approximate copies of a nucleotide sequence on the same strand or opposite strands. These are identified with FindSynteny, aligned with AlignSynteny, and then scored using the same statistical framework as tandem repeats.

The highest scoring repeat in each region is returned, unless allScores is TRUE, in which case overlapping repeats are permitted in the result.

Value

If type is "tandem", a data.frame giving the "Index" of the sequence in myXStringSet, "Begin" and "End" positions of tandem repeats, "Left" and "Right" positions of each repeat, and its "Score".

If type is "interspersed", a data.frame similar to the matrix in the lower diagonal of Synteny objects (see Synteny-class).

If type is "both", a list with the above two elements.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Cho, S.-T. & Wright E.S. (2025). "Accurate detection of tandem repeats exposes ubiquitous reuse of biological sequences." Nucleic Acids Research, **53(17)**, doi:10.1093/nar/gkaf866.

Schaper, E., et al. (2012). "Repeat or not repeat?-Statistical validation of tandem repeat prediction in genomic sequences." Nucleic Acids Research, **40(20)**, 10005-17.

See Also

ScoreAlignment

Run vignette("RepeatRepeat", package = "DECIPHER") to see a related vignette.

Examples

```
fas <- system.file("extdata", "Human_huntingtin_cds.fas.gz", package="DECIPHER")
dna <- readDNAStringSet(fas)
```

```
x <- DetectRepeats(dna)
x

# number of tandem repeats
lengths(x[, "Left"])

# average periodicity of tandem repeats
per <- mapply(function(a, b) b - a + 1,
 x[, "Left"],
 x[, "Right"],
 SIMPLIFY=FALSE)
sapply(per, mean)

# extract a tandem repeat
i <- 1
reps <- extractAt(dna[[x[i, "Index"]]],
 IRanges(x[[i, "Left"]], x[[i, "Right"]]))
reps
reps <- AlignSeqs(reps, verbose=FALSE) # align the repeats
reps
BrowseSeqs(reps)

# highlight tandem repeats in the sequence
colors <- c("deeppink", "deepskyblue")
colors <- lapply(colors, function(x) col2rgb(x)/255)
cols <- vector("list", length(dna))
for (i in seq_along(cols)) {
 cols[[i]] <- matrix(0, nrow=3, ncol=width(dna)[i])
 for (j in which(x[, "Index"] == i)) {
  left <- x[[j, "Left"]]
  right <- x[[j, "Right"]]
  n <- 0
  for (k in seq_along(left)) {
   r <- left[k]:right[k]
   n <- n + 1
   if (n > length(colors))
    n <- 1
   cols[[i]][, r] <- colors[[n]]
  }
 }
}
BrowseSeqs(dna, patterns=cols)

# find interspersed (rather than tandem) repeats
data(yeastSEQCHR1)
chr1 <- DNAStringSet(yeastSEQCHR1)

if (require("RSQLite", quietly=TRUE)) {
 z <- DetectRepeats(chr1, type="interspersed")
 z
}
```

DigestDNA *Simulate Restriction Digestion of DNA*

Description

Restriction enzymes can be used to cut double-stranded DNA into fragments at specific cut sites. DigestDNA performs an *in-silico* restriction digest of the input DNA sequence(s) given one or more restriction sites.

Usage

```
DigestDNA(sites,
          myDNAStringSet,
          type = "fragments",
          strand = "both")
```

Arguments

sites	A character vector of DNA recognition sequences and their enzymes' corresponding cut site(s).
myDNAStringSet	A DNAStringSet object or character vector with one or more sequences in 5' to 3' orientation.
type	Character string indicating the type of results desired. This should be either "fragments" or "positions".
strand	Character string indicating the strand(s) to cut. This should be one of "both", "top", or "bottom". The top strand is defined as the input DNAStringSet sequence, and the bottom strand is its reverse complement.

Details

In the context of a restriction digest experiment with a known (linear) DNA sequence, it can be useful to predict the expected DNA fragments *in-silico*. Restriction enzymes make cuts in double-stranded DNA at specific positions near their recognition site. The recognition site may be somewhat ambiguous, as represented by the IUPAC_CODE_MAP. Cuts that occur at different positions on the top and bottom strands result in sticky-ends, whereas those that occur at the same position result in fragments with blunt-ends. Multiple restriction sites can be supplied to simultaneously digest the DNA. In this case, sites for the different restriction enzymes may be overlapping, which could result in multiple close-proximity cuts that would not occur experimentally. Also, note that cut sites will not be matched to non-DNA_BASES in myDNAStringSet.

Value

DigestDNA can return two types of results: cut positions or the resulting DNA fragments corresponding to the top, bottom, or both strands. If type is "positions" then the output is a list with the cut location(s) in each sequence in myDNAStringSet. The cut location is defined as the position after the cut relative to the 5'-end. For example, a cut at 6 would occur between positions 5 and 6, where the respective strand's 5' nucleotide is defined as position 1.

If type is "fragments" (the default), then the result is a DNAStringSetList. Each element of the list contains the top and/or bottom strand fragments after digestion of myDNAStringSet, or the original sequence if no cuts were made. Sequences are named by whether they originated from the top or bottom strand, and list elements are named based on the input DNA sequences. The top strand is defined by myDNAStringSet as it is input, whereas the bottom strand is its reverse complement.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

DesignSignatures, RESTRICTION_ENZYMES

Examples

```
# digest hypothetical DNA sequences with BamHI
data(RESTRICTION_ENZYMES)
site <- RESTRICTION_ENZYMES[c("BamHI")]
dna <- DNAStringSet(c("AAGGATCCAA", "GGGATCAT"))
dna # top strand
reverseComplement(dna) # bottom strand
names(dna) <- c("hyp1", "hyp2")
d <- DigestDNA(site, dna)
d # fragments in a DNAStringSetList
unlist(d) # all fragments as one DNAStringSet

# Restriction digest of Yeast Chr. 1 with EcoRI and EcoRV
data(yeastSEQCHR1)
sites <- RESTRICTION_ENZYMES[c("EcoRI", "EcoRV")]
seqs <- DigestDNA(sites, yeastSEQCHR1)
seqs[[1]]

pos <- DigestDNA(sites, yeastSEQCHR1, type="positions")
str(pos)
```

Disambiguate *Expand Ambiguities into All Permutations of a DNAStringSet*

Description

Performs the inverse function of ConsensusSequence by expanding any ambiguities present in sequences.

Usage

```
Disambiguate(myXStringSet)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet object of sequences.

Details

Ambiguity codes in the IUPAC_CODE_MAP can be used to represent multiple nucleotides at a single position. Using these letters, multiple oligonucleotide permutations can be represented with a single ambiguous sequence. This function expands each sequence in the DNAStringSet input into all of its permutations. Note that sequences with many ambiguities can result in a very large number of potential permutations.

Value

A DNAStringSetList or RNAStringSetList with one element for each sequence in myXStringSet.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

ConsensusSequence

Examples

```
dna <- DNAStringSet(c("ACST", "NNN"))
dna_list <- Disambiguate(dna)
dna_list[[1]]
dna_list[[2]]
unlist(dna_list)

rna <- RNAStringSet(c("ACGU", "AGAU")) # 2 permutations
rna <- ConsensusSequence(rna) # "ASRU"
Disambiguate(rna) # 4 permutations
```

DistanceMatrix *Calculate the Distances Between Sequences*

Description

Calculates a distance matrix for an XStringSet. Each element of the distance matrix corresponds to the dissimilarity between two sequences in the XStringSet.

Usage

```
DistanceMatrix(myXStringSet,
               method = "overlap",
               type = "matrix",
               includeTerminalGaps = FALSE,
               penalizeGapLetterMatches = FALSE,
               minCoverage = 0,
               correction = NA,
               substitutionMatrix = NULL,
               frequencies = NULL,
               processors = 1,
               verbose = TRUE)
```

Arguments

myXStringSet	A DNAStringSet, RNAStringSet, or AAStringSet object of aligned sequences.
type	Character string indicating the type of output desired. This should be either "matrix" or "dist". (See value section below.)
method	Character string determining the region in which distance is calculated. This should be (an unambiguous abbreviation of) one of "overlap", "shortest", or "longest". The default method ("overlap") calculates distance in the overlapping region between terminal gaps when includeTerminalGaps is FALSE and the entire alignment otherwise. Setting method to "shortest" or "longest" will use the region between the start and end of the shortest or longest sequence, respectively, for each pairwise distance. The method is only relevant when includeTerminalGaps is TRUE.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" or "." characters on each end of the sequence) into the calculation of distance.
penalizeGapLetterMatches	Logical specifying whether to correct distance for gap-to-letter mismatches. The default (FALSE) ignores gaps paired with letters, and TRUE considers them as mismatches. If NA then gap-to-letter mismatches are only penalized once per run of gaps (i.e., insertion or deletion). If correction is not NA or frequencies is not NULL, the evolutionary distance due to insertion and deletion (Tajima, 1984) is added to the standard distance when penalizeGapLetterMatches is not FALSE.
minCoverage	Numeric giving the minimum fraction of sequence positions (not gap or mask) that must be overlapping in each pair. If positive then coverage is relative to the shortest sequence. If negative then coverage is relative to both sequences. Sequences failing to meet minCoverage will be assigned NA distances. Note that completely non-overlapping sequences are always given NA distances, regardless of minCoverage, unless includeTerminalGaps is TRUE and penalizeGapLetterMatches is not FALSE (i.e., distance = 100%).
correction	The evolutionary model used for distance correction. This should be either NA or either "JC69", "K80", "F81+F", "HKY85+F", "T92+F", "TN93+F", or one of the protein MODELS. The default (NA) will return uncorrected (i.e., length-normalized

Hamming) distances. Models with +F derive fixed frequencies empirically from myXStringSet (e.g., "F81+F" or "WAG+F") unless provided as frequencies. (See details section below.)

substitutionMatrix
: A symmetric matrix, dist object, or a single character string specifying a substitution matrix. Interpreted as costs if frequencies is NULL (the default), and interpreted as relative substitution rates otherwise. If a built-in substitutionMatrix is named (e.g., "BLOSUM62"), the negative of the matrix is used to obtain distances rather than similarities. Only used if correction is NA. (See examples section below.)

frequencies
: Numeric vector containing relative letter frequencies or NULL (the default) if unnecessary. Must be named and in the same order as substitutionMatrix. Only considered when substitutionMatrix is specified, in which case values are interpreted as stationary frequencies and substitutionMatrix must contain the corresponding substitution when correction is NA (the default).

processors
: The number of processors to use, or NULL to automatically detect and use all available processors.

verbose
: Logical indicating whether to display progress.

Details

DistanceMatrix computes pairwise distances between all sequences in myXStringSet. There are three main modes in which DistanceMatrix is typically used: (1) to obtain length-normalized Hamming distances (a.k.a., p-distances) with or without gap penalties, (2) to obtain phylogenetic distances by correcting for multiple substitutions per site based on a model of evolution, and (3) to obtain normalized distances based on a substitutionMatrix containing transition costs. For (1), distances are in units of differences/site, and the key parameters to consider are penalizeGapLetterMatches and method if includeTerminalGaps is TRUE. For (2), distances are in units of substitutions per site, and the key parameters are correction or substitutionMatrix and frequencies. For (3), distances are in the same units as the substitutionMatrix normalized by the number of shared sites, and substitutionMatrix is the main parameter to consider. These parameters are described in more detail below and several examples are provided in the examples section.

The defaults compute an uncorrected distance matrix representing the proportion of differences between each of the sequences in myXStringSet, also known as the p-distance. That is, matches are assigned a cost of 0 and mismatches are given a cost of 1, and distances is computed by dividing the number of mismatches by the number of matches plus mismatches. Ambiguity can be represented using the characters of the IUPAC_CODE_MAP for DNAStringSet and RNAStringSet inputs, or using the AMINO_ACID_CODE for an AAStringSet input. For example, the distance between an 'N' and any other nucleotide base is zero. The letters B (N or D), J (I or L), Z (Q or E), and X (any letter) are degenerate in the AMINO_ACID_CODE.

If includeTerminalGaps = FALSE then terminal gaps ("-" or "." characters) are not included in pairwise comparisons. This can be faster since only the positions common to each pair of sequences are compared. Sequences with no overlapping region in the alignment are given a value of NA, unless includeTerminalGaps = TRUE, in which case distance is 100%. Both "-" and "." characters are interpreted as gaps, and masked characters ("+") in either sequence are not considered in distance. The default behavior is to calculate distance as the fraction of positions that differ across the region of the alignment shared by both sequences (not including gaps). Gap-to-letter matches are treated as

mismatches if penalizeGapLetterMatches is TRUE. Only the first gap per run of gaps is penalized if penalizeGapLetterMatches is NA, which represents an event based model where insertions and deletions are considered as single events analogous to individual substitutions (also known as a block model).

Multiple correction factors are available to transform distances into an expected number of changes per site. There are six models specifically for nucleotides ("JC69", "F81+F", "K80", "HKY85+F", "T92+F", and "TN93+F") that introduce different transition (versus transversion) rates. The analytical formula for "JC69" gives the exact distance, while the others use approximation formulas to circumvent parameter estimation (McGuire, 1999; Tamura, 1993). In order of decreasing exactness they are: "K80", "F81+F", "HKY85+F", "TN93+F", and "T92+F". Models with fixed frequencies (+F) derive empirical frequencies from the input myXStringSet, unless they are given as frequencies.

The remaining corrections are defined by MODELS of amino acid evolution that provide both a matrix of substitution rates and stationary frequencies. These models can be modified with empirical frequencies (+F), or frequencies can be given separately. It is also feasible to specify the substitutionMatrix (rates) and frequencies separately when correction is NA (the default). A maximum likelihood distance is returned based on fitting

$$P = e^(Q * d)$$

, where P is the observed substitution probabilities and Q is derived from substitutionMatrix such that

$$Q * frequencies = 0$$

. Here, d represents twice the estimated number of substitutions per sites that occurred since a pair of sequences most recent common ancestor, i.e., their phylogenetic distance.

If penalizeGapLetterMatches is not FALSE when using a model of evolution, the component of distance due to insertion and deletion is added to the distance (Tajima, 1984). That is, the traditional evolutionary distance due to multiple substitutions is added to

$$-2 * \log\left(n_x y / \left(n_x + n_y\right)^0 .5\right)$$

, where $n_x y$ is the number of shared sites, n_x is the number of sites in one sequence and n_y is the number in the other. This partly corrects for multiple insertion and deletion events in a related manner to correcting for multiple substitutions per site. The resulting distances can be considered to have units of changes per site, since substitutions, insertions, and deletions all contribute to the distance.

If a substitutionMatrix is given without frequencies, the matrix is interpreted as the cost for each type of substitution. It is also possible to specify substitutionMatrix as a character string (e.g., "BLOSUM62"), and the negative of the built-in substitution matrix is used to obtain distances rather than similarities. This is useful for obtaining pairwise substitution scores according to a cost matrix.

Value

If type is "matrix", a symmetric matrix where each element is the distance between the sequences referenced by the respective row and column. The dimnames of the matrix correspond to the names of the XStringSet.

If type is "dist", an object of class "dist" that contains the lower triangle of the distance matrix as a vector. Since the distance matrix is symmetric, storing only one triangle is more memory efficient.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

McGuire G., *et al.* (1999) Improved error bounds for genetic distances from DNA sequences. *Biometrics*, **55(4)**, 1064-1070.

Tajima, F. and Nei, M. (1984) Estimation of evolutionary distance between nucleotide sequences. *Molecular Biology and Evolution*, **1(3)**, 269-285.

Tamura, K. and Nei, M. (1993) Estimation of the number of nucleotide substitutions in the control region of mitrochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution*, **10(3)**, 512-526.

Wright, E. (2026) Treeline Provides a Unified Strategy for Optimising Phylogenetic Trees Under Alternative Criteria. *Molecular Ecology Resources*, **26(5)**, e70163.

See Also

MODELS, Treeline

Run `vignette("GrowingTrees", package = "DECIPHER")` to see a related vignette.

Examples

```
# example of using the defaults
dna <- DNAStringSet(c("ACTG", "ACCG"))
dna
DistanceMatrix(dna)

# changing the output type to "dist"
d <- DistanceMatrix(dna, type="dist")
d
length(d) # minimal memory space required
m <- as.matrix(d)
length(m) # more memory space required

# supplying an AAStringSet
aa <- AAStringSet(c("ASYK", "ATYK", "CTWN"))
aa
DistanceMatrix(aa)

# correct for multiple substitutions per site
DistanceMatrix(aa, correction="Poisson")

# defaults compare intersection of internal ranges
dna <- DNAStringSet(c("ANGCT-", "-ACCT-"))
dna
d <- DistanceMatrix(dna)
d
# d[1,2] is 1 base in 4 = 0.25

# compare union of internal positions, without terminal gaps
```

```
dna <- DNAStringSet(c("ANGCT-", "-ACCT-"))
dna
d <- DistanceMatrix(dna,
                    includeTerminalGaps=TRUE,
                    penalizeGapLetterMatches=TRUE)
d
# d[1,2] is now 2 bases in 5 = 0.40

# gap ("-") and unknown (".") characters are interchangeable
dna <- DNAStringSet(c("ANGCT.", ".ACCT-"))
dna
d <- DistanceMatrix(dna,
                    includeTerminalGaps=TRUE,
                    penalizeGapLetterMatches=TRUE)
d
# d[1,2] is still 2 bases in 5 = 0.40

# compare different methods for calculating distance
dna <- DNAStringSet(c("--ACTG", "TGAGT-"))
dna
DistanceMatrix(dna, method="overlap") # 1/3
DistanceMatrix(dna, method="shortest",
               includeTerminalGaps=FALSE) # 1/3
DistanceMatrix(dna, method="shortest",
               includeTerminalGaps=TRUE,
               penalizeGapLetterMatches=TRUE) # 2/4
DistanceMatrix(dna, method="shortest",
               includeTerminalGaps=TRUE) # 1/3
DistanceMatrix(dna, method="longest",
               includeTerminalGaps=FALSE) # 1/3
DistanceMatrix(dna, method="longest",
               includeTerminalGaps=TRUE,
               penalizeGapLetterMatches=TRUE) # 3/5
DistanceMatrix(dna, method="longest",
               includeTerminalGaps=TRUE) # 1/3
DistanceMatrix(dna, method="overlap",
               minCoverage=1) # NA (insufficient overlap)
DistanceMatrix(dna, method="overlap",
               minCoverage=0.75) # 3/4 sites covered in shorter
DistanceMatrix(dna, method="overlap",
               minCoverage=-0.75) # 3/5 sites covered in longer

# neither internal nor external gap/gap matches are considered
dna <- DNAStringSet(c("--A-CTA", "-AG-C--"))
dna
DistanceMatrix(dna) # 1/2
DistanceMatrix(dna,
               includeTerminalGaps=TRUE,
               penalizeGapLetterMatches=TRUE) # 4/5
DistanceMatrix(dna,
               includeTerminalGaps=TRUE,
               penalizeGapLetterMatches=TRUE,
               method="shortest") # 2/3
```

```
DistanceMatrix(dna,
               includeTerminalGaps=TRUE,
               penalizeGapLetterMatches=TRUE,
               method="longest") # 3/4

# examples using real sequences
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
ali <- AlignTranslation(dna, type="both", verbose=FALSE)
DNA <- ali[[1L]]
AA <- ali[[2L]]

# p-distances with different gap penalties
d_NT_noGap <- DistanceMatrix(DNA, type="dist") # penalizeGapLetter=FALSE
d_NT_allGap <- DistanceMatrix(DNA, type="dist", penalizeGapLetter=TRUE)
d_NT_oneGap <- DistanceMatrix(DNA, type="dist", penalizeGapLetter=NA)
pairs(data.frame(d_NT_noGap, d_NT_allGap, d_NT_oneGap), pch=46, col="#00000011")

# correcting distances for multiple substitutions per site
Q <- 1 - diag(4)
rownames(Q) <- colnames(Q) <- DNA_BASES
freqs <- setNames(rep(0.25, 4), DNA_BASES)
d_NT_JC <- DistanceMatrix(DNA, type="dist", correction="JC")
d_NT_even <- DistanceMatrix(DNA, type="dist", substitutionMatrix=Q, frequencies=freqs)
pairs(data.frame(d_NT_noGap, d_NT_JC, d_NT_even), pch=46, col="#00000011")

# specifying a rate matrix with uneven transition/transversion rates
Q["C", "T"] <- Q["T", "C"] <- Q["A", "G"] <- Q["G", "A"] <- 2
d_NT_uneven <- DistanceMatrix(DNA, type="dist", substitutionMatrix=Q, frequencies=freqs)
pairs(data.frame(d_NT_noGap, d_NT_uneven, d_NT_even), pch=46, col="#00000011")

# comparing amino acid and nucleotide distances
d_AA_noGap <- DistanceMatrix(AA, type="dist") # penalizeGapLetter=FALSE
d_AA_WAG <- DistanceMatrix(AA, type="dist", correction="WAG")
pairs(data.frame(d_NT_noGap, d_AA_noGap, d_AA_WAG), pch=46, col="#00000011")

# using a model with letter frequencies derived from the input sequences
d_NT_F81_F <- DistanceMatrix(DNA, type="dist", correction="F81+F")
d_AA_WAG_F <- DistanceMatrix(AA, type="dist", correction="WAG+F")
pairs(data.frame(d_NT_F81_F, d_AA_WAG, d_AA_WAG_F), pch=46, col="#00000011")

# choosing a model with different transition/transversion rates
d_NT_K80 <- DistanceMatrix(DNA, type="dist", correction="K80")
d_NT_TN93_F <- DistanceMatrix(DNA, type="dist", correction="TN93+F")
pairs(data.frame(d_NT_F81_F, d_NT_K80, d_NT_TN93_F), pch=46, col="#00000011")

# incorporating the indel component of distance
d_NT_JC_indel <- DistanceMatrix(DNA, type="dist", correction="JC", penalize=TRUE)
d_NT_JC_indelblock <- DistanceMatrix(DNA, type="dist", correction="JC", penalize=NA)
pairs(data.frame(d_NT_JC, d_NT_JC_indel, d_NT_JC_indelblock), pch=46, col="#00000011")

# specifying a substitutionMatrix (without frequencies)
d_AA_BLOSUM <- DistanceMatrix(AA, type="dist", substitutionMatrix="BLOSUM62")
```

```
d_AA_PFASUM <- DistanceMatrix(AA, type="dist", substitutionMatrix="PFASUM50")
pairs(data.frame(d_AA_WAG, d_AA_BLOSUM, d_AA_PFASUM), pch=46, col="#00000011")
```

ExtractGenes *Extract Predicted Genes from a Genome*

Description

Extracts predicted genes from the genome used for prediction.

Usage

```
ExtractGenes(x,
             myDNAStringSet,
             type = "DNAStringSet",
             ...)
```

Arguments

x	An object of class Genes.
myDNAStringSet	The DNAStringSet object used in generating x.
type	The class of sequences to return. This should be (an unambiguous abbreviation of) one of "AAStringSet", "DNAStringSet" (the default), or "RNAStringSet".
...	Other parameters passed directly to translate.

Details

Extracts a set of gene predictions as either DNA, mRNA, or proteins.

Value

An "AAStringSet", "DNAStringSet", or "RNAStringSet" determined by type.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

FindGenes, Genes-class, WriteGenes

Examples

```
# import a test genome
fas <- system.file("extdata",
 "Chlamydia_trachomatis_NC_000117.fas.gz",
 package="DECIPHER")
genome <- readDNAStringSet(fas)

x <- FindGenes(genome)
genes <- ExtractGenes(x, genome)
proteins <- ExtractGenes(x, genome, type="AAStringSet")
```

FindChimeras *Find Chimeras in a Sequence Database*

Description

Finds chimeras present in a database of sequences. Makes use of a reference database of (presumed to be) good quality sequences.

Usage

```
FindChimeras(dbFile,
             tblName = "Seqs",
             identifier = "",
             dbFileReference,
             tblNameReference = "Seqs",
             batchSize = 100,
             minNumFragments = 20000,
             tb.width = 5,
             multiplier = 20,
             minLength = 30,
             minCoverage = 0.6,
             overlap = 100,
             minSuspectFragments = 4,
             showPercentCoverage = FALSE,
             add2tbl = FALSE,
             maxGroupSize = -1,
             minGroupSize = 25,
             excludeIDs = NULL,
             processors = 1,
             verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file to be checked for chimeric sequences.
tblName	Character string specifying the table in which to check for chimeras.

identifier Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected.

dbFileReference
A database connection object or a character string specifying the path to a SQLite reference database file of (presumed to be) good quality sequences.

tblNameReference
Character string specifying the table with reference sequences.

batchSize Number sequences to tile with fragments at a time.

minNumFragments
Number of suspect fragments to accumulate before searching through other groups.

tb.width A single integer ([1..14]) giving the number of nucleotides at the start of each fragment that are part of the trusted band.

multiplier A single integer specifying the multiple of fragments found out-of-group greater than fragments found in-group in order to consider a sequence a chimera.

minLength Minimum length of a chimeric region in order to be considered as a chimera.

minCoverage Minimum fraction of coverage necessary in a chimeric region.

overlap Number of nucleotides at the end of the sequence that the chimeric region must overlap in order to be considered a chimera.

minSuspectFragments
Minimum number of suspect fragments belonging to another group required to consider a sequence a chimera.

showPercentCoverage
Logical indicating whether to list the percent coverage of suspect fragments in each chimeric region in the output.

add2tbl Logical or a character string specifying the table name in which to add the result.

maxGroupSize Maximum number of sequences searched in a group. A value of less than 0 means the search is unlimited.

minGroupSize The minimum number of sequences in a group to be considered as part of the search for chimeras. May need to be set to a small value for reference databases with mostly small groups.

excludeIDs Optional character vector of identifier(s) to exclude from database searches, or NULL (the default) to not exclude any.

processors The number of processors to use, or NULL to automatically detect and use all available processors.

verbose Logical indicating whether to display progress.

Details

FindChimeras works by finding suspect k-mer fragments that are uncommon in the group where the sequence belongs, but very common in another group where the sequence does not belong. Each sequence in the dbFile is tiled into short sequence segments called fragments. If the fragments are infrequent in their respective group in the dbFileReference then they are considered suspect.

If enough suspect fragments from a sequence meet the specified constraints then the sequence is flagged as a chimera.

The default parameters are optimized for full-length 16S sequences (> 1,000 nucleotides). Shorter 16S sequences require two parameters that are different than the defaults: `minCoverage = 0.2`, and `minSuspectFragments = 2`.

Groups are determined by the identifier present in each database. For this reason, the groups in the `dbFile` should exist in the groups of the `dbFileReference`. The reference database is assumed to contain many sequences of only good quality.

If a reference database is not present then it is feasible to create a reference database by using the input database as the reference database. Removing chimeras from the reference database and then iteratively repeating the process can result in a clean reference database.

For non-16S sequences it may be necessary to optimize the parameters for the particular sequences. The simplest way to perform an optimization is to experiment with different input parameters on artificial chimeras such as those created using `CreateChimeras`. Adjusting input parameters until the maximum number of artificial chimeras are identified is the easiest way to determine new defaults.

Value

A `data.frame` containing only the sequences that meet the specifications for being chimeric. The chimera column contains information on the chimeric region and to which group it belongs. The `row.names` of the `data.frame` correspond to those of the sequences in the `dbFile`.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright et al. (2012) "DECIPHER: A Search-Based Approach to Chimera Identification for 16S rRNA Sequences." Applied and Environmental Microbiology, doi:10.1128/AEM.06516-11.

See Also

`CreateChimeras`, `Add2DB`

Run `vignette("FindChimeras", package = "DECIPHER")` to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 # It is necessary to set dbFileReference to the file path of the
 # 16S reference database available from http://DECIPHER.codes
 chimeras <- FindChimeras(db, dbFileReference=db)
}
```

FindGenes *Find Genes in a Genome*

Description

Predicts the start and stop positions of protein coding genes in a genome.

Usage

```
FindGenes(myDNAStringSet,
          geneticCode = getGeneticCode("11"),
          minGeneLength = 60,
          includeGenes = NULL,
          allowEdges = TRUE,
          allScores = FALSE,
          showPlot = FALSE,
          processors = 1,
          verbose = TRUE)
```

Arguments

myDNAStringSet	A DNAStringSet object of unaligned sequences representing a genome.
geneticCode	A named character vector defining the translation from codons to amino acids. Optionally, an "alt_init_codons" attribute can be used to specify alternative initiation codons. By default, the bacterial and archaeal genetic code is used, which has seven possible initiation codons: ATG, GTG, TTG, CTG, ATA, ATT, and ATC.
minGeneLength	Integer specifying the minimum length of genes to find in the genome.
includeGenes	A Genes object to include as potential genes or NULL (the default) to predict all genes de novo.
allowEdges	Logical determining whether to allow genes that run off the edge of the sequences. If TRUE (the default), genes can be identified with implied starts or ends outside the boundaries of myDNAStringSet, although the boundary will be set to the last possible codon position.
allScores	Logical indicating whether to return information about all possible open reading frames or only the predicted genes (the default).
showPlot	Logical determining whether a plot is displayed with the distribution of gene lengths and scores. (See details section below.)
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to print information about the predictions on each iteration. (See details section below.)

Details

Protein coding genes are identified by learning their characteristic signature directly from the genome, i.e., *ab initio* prediction. Gene signatures are derived from the content of the open reading frame and surrounding signals that indicate the presence of a gene. Genes are assumed to not contain introns or frame shifts, making the function best suited for prokaryotic genomes. Partial genes at contig borders can be identified in incomplete or circular genomes if `allowEdges` is `TRUE` (the default).

If `showPlot` is `TRUE` then a plot is displayed with four panels. The upper left panel shows the fitted distribution of background open reading frame lengths. The upper right panel shows this distribution on top of the fitted distribution of predicted gene lengths. The lower left panel shows the fitted distribution of scores for the intergenic spacing between genes on the same and opposite genome strands. The bottom right panel shows the total score of open reading frames and predicted genes by length.

If `verbose` is `TRUE`, information is shown about the predictions at each iteration of gene finding. The mean score difference between genes and non-genes is updated at each iteration, unless it is negative, in which case the score is dropped and a `"-"` is displayed. The columns denote the number of iterations (`"Iter"`), number of codon scoring models (`"Models"`), start codon scores (`"Start"`), upstream k-mer motif scores (`"Motif"`), mRNA folding scores (`"Fold"`), initial codon bias scores (`"Init"`), upstream nucleotide bias scores (`"UpsNt"`), termination codon bias scores (`"Term"`), ribosome binding site scores (`"RBS"`), codon autocorrelation scores (`"Auto"`), stop codon scores (`"Stop"`), and number of predicted genes (`"Genes"`).

Value

An object of class `Genes`.

Author(s)

Erik Wright `<Erik.Wright@bcm.edu>`

See Also

`ExtractGenes`, `FindNonCoding`, `Genes-class`, `WriteGenes`

Run `vignette("FindingGenes", package = "DECIPHER")` to see a related vignette.

Examples

```
# import a test genome
fas <- system.file("extdata",
 "Chlamydia_trachomatis_NC_000117.fas.gz",
 package="DECIPHER")
genome <- readDNAStringSet(fas)

z <- FindGenes(genome)
z
genes <- ExtractGenes(z, genome)
genes
proteins <- ExtractGenes(z, genome, type="AAStringSet")
proteins
```

FindNonCoding *Find Non-Coding RNAs in a Genome*

Description

Searches for conserved patterns representing a family of non-coding RNAs. Returns the start and end boundaries of potential matches along with their log-odds score.

Usage

```
FindNonCoding(x,
              myXStringSet,
              minScore = 16,
              allScores = FALSE,
              processors = 1,
              verbose = TRUE)
```

Arguments

x	A NonCoding object or a list of NonCoding objects for searching.
myXStringSet	A DNAStringSet or RNAStringSet object of unaligned sequences, typically representing a genome.
minScore	Numeric giving the minimum log-odds score of matches to x in myXStringSet to report, or a vector of numerics specifying the minimum score per NonCoding object in x. The maximum false discovery rate is approximately exp(-minScore) per nucleotide per object in x.
allScores	Logical specifying whether all matches should be returned (TRUE) or only the top matches when there are multiple matches in the same region.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

Non-coding RNAs are identified by the location of representative sequence patterns relative to the beginning and end of the non-coding RNA. Potential matches to each NonCoding object in x are scored based on their log-odds relative to a background that is derived from the input sequence (myXStringSet). Matches of at least minScore are returned as a Genes object with the "Gene" column set to the negative index of the list element of x that was identified.

Value

An object of class Genes.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2021). FindNonCoding: rapid and simple detection of non-coding RNAs in genomes. Bioinformatics. https://doi.org/10.1093/bioinformatics/btab708

See Also

LearnNonCoding, NonCoding-class, ExtractGenes, Genes-class, WriteGenes

Run `vignette("FindingNonCodingRNAs", package = "DECIPHER")` to see a related vignette.

Examples

```
data(NonCodingRNA_Bacteria)
x <- NonCodingRNA_Bacteria
names(x)

# import a test genome
fas <- system.file("extdata",
 "Chlamydia_trachomatis_NC_000117.fas.gz",
 package="DECIPHER")
genome <- readDNAStringSet(fas)

z <- FindNonCoding(x, genome)
z

annotations <- attr(z, "annotations")
m <- match(z[, "Gene"], annotations)
sort(table(names(annotations)[m]))

genes <- ExtractGenes(z, genome, type="RNAStringSet")
genes
```

FindSynteny *Finds Synteny in a Sequence Database*

Description

Finds syntenic blocks between groups of sequences in a database.

Usage

```
FindSynteny(dbFile,
            tblName = "Seqs",
            identifier = "",
            useFrames = TRUE,
            alphabet = AA_REDUCED[[172]],
            geneticCode = GENETIC_CODE,
            sepCost = -1,
            sepPower = 0.5,
```

```
gapCost = -9,
gapPower = 0.5,
shiftCost = 0,
codingCost = 0,
maxSep = 600,
maxGap = 400,
minScore = 100,
N = 10,
dropScore = -5,
maskRepeats = TRUE,
maskLCRs = TRUE,
allowOverlap = FALSE,
storage = 0.5,
processors = 1,
verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file. The database should contain DNA sequences, typically with a distinct identifier for sequences belonging to each genome or chromosome.
tblName	Character string specifying the table where the sequences are located.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected. Repeated identifiers will find synteny between a sequence and itself, while blocking identical positions from matching in both sequences.
useFrames	Logical specifying whether to use 6-frame amino acid translations to help find more distant hits. Using the alphabet is helpful when the genome is largely composed of coding DNA. If FALSE then faster but less sensitive to distant homology.
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between sequences. A non-reduced amino acid alphabet can be used by setting alphabet equal to AA_STANDARD.
geneticCode	Either a character vector giving the genetic code to use in translation, or a list containing one genetic code for each identifier. If a list is provided then it must be named by the corresponding identifiers in the database.
sepCost	Cost per nucleotide separation between hits to apply when chaining hits into blocks.
sepPower	Positive numeric specifying the power applied to the separation between hits before multiplying by sepCost.
gapCost	Cost for gaps between hits to apply when chaining hits into blocks.
gapPower	Positive numeric specifying the power applied to the number of gaps between hits before multiplying by gapCost.

shiftCost	Cost for shifting between different reading frames when chaining reduced amino acid hits into blocks.
codingCost	Cost for switching between coding and non-coding hits when chaining hits into blocks.
maxSep	Maximal separation (in nucleotides) between hits in the same block.
maxGap	The maximum number of gaps between hits in the same block.
minScore	The minimum score required for a chain of hits to become a block. Higher values of minScore are less likely to yield false positives.
N	Numeric indicating the approximate number of k-mers that can be randomly selected before one is found by chance on average. For example, the default value of 10 will set k-mer length such that every 10th k-mer is expected to have a match by chance.
dropScore	The change from maximal score required to stop extending blocks.
maskRepeats	Logical specifying whether to mask repeats when searching for hits.
maskLCRs	Logical indicating whether to mask low complexity regions when searching for hits.
allowOverlap	Logical specifying whether to permit blocks to overlap on the same sequence.
storage	Excess gigabytes available to store objects so that they do not need to be re-computed in later steps. This should be a number between zero and a (modest) fraction of the available system memory. Note that more than storage gigabytes may be required, but will not be stored for later reuse.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

Long nucleotide sequences, such as genomes, are often not collinear or may be composed of many smaller segments (e.g., contigs). FindSynteny searches for "hits" between sequences that can be chained into collinear "blocks" of synteny. Hits are defined as k-mer exact nucleotide matches or k-mer matches in a reduced amino acid alphabet (if useFrames is TRUE). Hits are chained into blocks as long as they are: (1) within the same sequence, (2) within maxSep and maxGap distance, and (3) help maintain the score above minScore. Blocks are extended from their first and last hit until their score drops below dropScore from the maximum that was reached. This process results in a set of hits and blocks stored in an object of class "Synteny".

Value

An object of class "Synteny".

Note

FindSynteny is intended to be used on sets of sequences with up to ~200 million nucleotides total per identifier. For this reason, better performance can sometimes be achieved by assigning a unique identifier to each chromosome belonging to a large genome.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

AlignSynteny, Synteny-class

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
 synteny <- FindSynteny(db)
 synteny
 pairs(synteny) # scatterplot matrix
}
```

FormGroups *Forms Groups By Rank*

Description

Agglomerates sequences into groups within a specified size range based on taxonomic rank.

Usage

```
FormGroups(dbFile,
           tblName = "Seqs",
           goalSize = 50,
           minGroupSize = 25,
           maxGroupSize = 5000,
           includeNames = FALSE,
           add2tbl = FALSE,
           verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the taxonomic rank (i.e., "organism") information is located.
goalSize	Number of sequences required in each group to stop adding more sequences.
minGroupSize	Minimum number of sequences in each group required to stop trying to recombine with a larger group.
maxGroupSize	Maximum number of sequences in each group allowed to continue agglomeration.

includeNames	Logical indicating whether to include the formal scientific name in the group name.
add2tbl	Logical or a character string specifying the table name in which to add the result.
verbose	Logical indicating whether to display progress.

Details

FormGroups uses the "organism" field in the dbFile table to group sequences with similar taxonomic rank. Taxonomic rank information must be present in the tblName, such as that created by default when importing sequences from a GenBank formatted file.

Organism information contains the formal scientific name on the first line, followed by the taxonomic lineage on subsequent lines. When includeNames is TRUE the formal scientific name is appended to the end of the group name, otherwise only the taxonomic lineage is used as the group name.

The algorithm ascends the taxonomic tree, agglomerating taxa into groups until the goalSize is reached. If the group size is below minGroupSize then further agglomeration is attempted with a larger group. If additional agglomeration results in a group larger than maxGroupSize then the agglomeration is undone so that the group is smaller. Setting minGroupSize to goalSize avoids the creation of polyphyletic groups. Note that this approach may often result in paraphyletic groups.

Value

A data.frame with the organism and corresponding group name as identifier. Note that quotes are stripped from group names to prevent problems that they may cause. The origin gives the organism preceding the identifier. The count denotes number of sequences corresponding to each organism. If add2tbl is not FALSE then the "identifier" and "origin" columns are updated in dbFile.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

IdentifyByRank

Run vignette("FindChimeras", package = "DECIPHER") to see a related vignette.

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 g <- FormGroups(db, goalSize=10, minGroupSize=5, maxGroupSize=20)
 head(g)
 tapply(g$count, g$identifier, sum)
}
```

Genes *Genes objects and accessors*

Description

Gene prediction consist of delimiting the boundaries of regions that function as genes within a genome. Class Genes provides objects and functions for storing the boundaries of genes and associated information resulting from gene prediction.

Usage

```
## S3 method for class 'Genes'
plot(x,
    xlim = c(1, 1e4),
    ylim = c(-100, 500),
    interact = FALSE,
    colorBy="Strand",
    colorRamp=colorRampPalette(c("darkblue", "darkred")),
    colorGenes="green4",
    ...)

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

## S3 method for class 'Genes'
x[i, j, ...]
```

Arguments

x	An object of class Genes.
xlim	Numeric vector of length 2 specifying the x-axis limits for plotting.
ylim	Numeric vector of length 2 specifying the y-axis limits for plotting.
interact	Logical determining whether the plot is interactive. If TRUE, clicking the plot on the right or left side will scroll one frame in that direction. To end interaction, either right-click, press the escape key, or press the stop button depending on the graphics device in use.
colorBy	Character string indicating the name of the column in x that should be used for coloring. Unambiguous abbreviations are also permitted.
colorRamp	A function that will return n colors when given a number n. Examples are rainbow, heat.colors, terrain.colors, cm.colors, or (the default) colorRampPalette.
colorGenes	Character string specifying the color of genes, or NA to color genes according to colorBy.
i	Numeric or character vector of row indices to extract from x.
j	Numeric or character vector of column indices to extract from x. If j is missing, all columns are included and the returned object will also belong to class Genes.
...	Other optional parameters.

Details

Objects of class Genes are stored as numeric matrices containing information pertaining to gene predictions. The matrix columns include the index ("Index") of the corresponding sequence in the original genome, the strand ("Strand") where the gene is located (either "+" (0) or "-" (1), the beginning ("Begin") and ending ("End") positions of the gene, scores acquired during prediction, and whether (!= 0) or not (0) the region was predicted to be a gene. Note that the start of the gene is at the beginning position when the strand is "+" and end when the strand is "-". By convention, rows with negative values in the "Gene" column represent non-coding RNAs and rows with positive values represent protein coding genes.

The plot method will show the total score of each prediction along the genome. This is most useful when displaying the result of setting allScores to TRUE in FindGenes. Here, possible genes on either strand will be shown (by default), with the predicted genes highlighted. The beginning (solid) and ending (dashed) positions are denoted by vertical lines. Note that the x-axis is cumulative genome position, and changes between genome sequences indices are demarcated by dashed vertical lines.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

ExtractGenes, FindGenes, WriteGenes

Examples

```
# import a test genome
fas <- system.file("extdata",
 "Chlamydia_trachomatis_NC_000117.fas.gz",
 package="DECIPHER")
genome <- readDNAStringSet(fas)

x <- FindGenes(genome, allScores=TRUE)
x
head(unclass(x)) # the underlying structure

plot(x) # default coloring by "Strand"
# color by RBS score (blue is weak/low, red is strong/high)
plot(x, colorBy="RibosomeBindingSiteScore", colorGenes=NA)
# color by fraction of times a gene was chosen
plot(x, colorBy="FractionReps", colorGenes=NA)
# color by which codon model was selected for each ORF
plot(x, colorBy="CodonModel", xlim=c(1, 3e4))

# example of splitting a Genes object by "Strand"
tapply(seq_len(nrow(x)), x[, "Strand"], function(i) x[i])
```

HEC_MI *Mutual Information for Protein Secondary Structure Prediction*

Description

Arrays containing values of mutual information for single residues (HEC_MI1) and pairs of residues (HEC_MI2) located within 10 residues of the position being predicted (position "0"). The arrays have dimensions corresponding to the 20 (standard) amino acids, positions (-10 to 10), and states (helix ("H"), sheet ("E"), or coil ("C")).

Usage

```
data("HEC_MI1")
data("HEC_MI2")
```

Format

The format of HEC_MI1 is: num [1:20, 1:21, 1:3] 0.04264 -0.00117 0.02641 0.08264 -0.04876 ... - attr(*, "dimnames")=List of 3 ..$: chr [1:20] "A" "R" "N" "D" ... ..$: chr [1:21] "-10" "-9" "-8" "-7" ... ..$: chr [1:3] "H" "E" "C"

The format of HEC_MI2 is: num [1:20, 1:20, 1:21, 1:21, 1:3] 2.56 -Inf -Inf -Inf -Inf ... - attr(*, "dimnames")=List of 5 ..$: chr [1:20] "A" "R" "N" "D" ... ..$: chr [1:20] "A" "R" "N" "D" ... ..$: chr [1:21] "-10" "-9" "-8" "-7" ... ..$: chr [1:21] "-10" "-9" "-8" "-7" ... ..$: chr [1:3] "H" "E" "C"

Details

The values in each matrix were derived based on a set of 15,201 proteins in the ASTRAL Compendium (Chandonia, 2004). The 8-states assigned by the Dictionary of Protein Secondary Structure (DSSP) were reduced to 3-states via H = G, H, or I; E = E; and C = B, S, C, or T.

References

Chandonia, J. M. (2004). The ASTRAL Compendium in 2004. *Nucleic Acids Research*, **32(90001)**, 189D-192. doi:10.1093/nar/gkh034.

Examples

```
data(HEC_MI1)
# the contribution of an arginine ("R")
# located 3 residues left of center
# to a helical ("H") state at the center
HEC_MI1["R", "-3", "H"]

data(HEC_MI2)
# the contribution of arginine and lysine ("K")
# located at positions -1 and +1, respectively
# to a coil ("C") state at the center position
HEC_MI2["R", "K", "-1", "1", "C"]
```

```
matplot(-10:10, t(HEC_MI1[,, "H"]),
       type="l", col=1:8, lty=rep(1:3, each=8),
       xlab="Amino Acid Position Relative to Center",
       ylab="Log-Odds of Helix at Center Position")
legend("bottomleft",
       lwd=1, col=1:8, lty=rep(1:3, each=8),
       legend=dimnames(HEC_MI1)[[1]], ncol=2)
```

IdConsensus *Create Consensus Sequences by Groups*

Description

Forms a consensus sequence representing the sequences in each group.

Usage

```
IdConsensus(dbFile,
            tblName = "Seqs",
            identifier = "",
            type = "DNAStringSet",
            colName = "identifier",
            processors = 1,
            verbose = TRUE,
            ...)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table in which to form consensus.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected.
type	The type of XStringSet (sequences) to use in forming consensus. This should be one of "DNAStringSet", "RNAStringSet", "AAStringSet", or "BStringSet".
colName	Column containing the group name of each sequence.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Additional arguments to be passed directly to ConsensusSequence for an AAStringSet, DNAStringSet, or RNAStringSet, or to consensusString for a BStringSet.

Details

Creates a consensus sequence for each of the distinct groups defined in `colName`. The resulting `XStringSet` contains as many consensus sequences as there are distinct groups in `colName`. For example, it is possible to create a set of consensus sequences with one consensus sequence for each `"id"` in the `tblName`.

Value

An `XStringSet` object containing the consensus sequence for each group. The `names` of the `XStringSet` contain the number of sequences and name of each group.

Author(s)

Erik Wright `<Erik.Wright@bcm.edu>`

See Also

`Seqs2DB`

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 con <- IdConsensus(db, colName="identifier", noConsensusChar="N")
 BrowseSeqs(con)
}
```

IdentifyByRank *Identify By Taxonomic Rank*

Description

Identifies sequences by a specific level of their taxonomic rank.

Usage

```
IdentifyByRank(dbFile,
               tblName = "Seqs",
               level = 0,
               add2tbl = FALSE,
               verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the taxonomic rank (i.e., "organism") information is located.
level	Level of the taxonomic rank. (See details section below.)
add2tbl	Logical or a character string specifying the table name in which to add the result.
verbose	Logical indicating whether to print database queries and other information.

Details

IdentifyByRank simply identifies a sequence by a specific level of its taxonomic rank. Requires that organism information be present in the tblName, such as that created by default when importing sequences from a GenBank formatted file.

The input parameter level should be an integer giving the "level" of the taxonomic rank to choose as the identifier. Negative levels are interpreted as being that many levels from the last level in each rank. The level zero selects the base level (see below).

If the specified level of rank does not exist then the closest rank is chosen. Therefore, setting level to Inf will always select the last taxonomic level (i.e., genus).

For example, a representative "organism" imported from a GenBank file is:
Saccharomyces cerevisiae
Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
Saccharomycetales; Saccharomycetaceae; Saccharomyces.

Setting level to 0 would result in an identifier of "Saccharomyces cerevisiae", because it is on the first line. A level of 2 would return "Fungi", and -2 (second to last) would return "Saccharomycetaceae". A level of Inf would find the nearest level to the end, "Saccharomyces".

Value

A data.frame with the organism and corresponding identifier as identifier. Note that quotes are stripped from identifiers to prevent problems that they may cause. The origin gives the organism preceding the identifier. If add2tbl is not FALSE then the "identifier" column is updated in dbFile.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

FormGroups

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 ids <- IdentifyByRank(db, level=Inf)
 head(ids)
}
```

IdLengths *Determine the Number of Characters in Each Sequence*

Description

Counts the number of standard and non-standard characters in each sequence.

Usage

```
IdLengths(dbFile,
          tblName = "Seqs",
          type = "DNAStringSet",
          add2tbl = FALSE,
          batchSize = 10000,
          processors = 1,
          verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the sequences are located.
type	The type of XStringSet being processed. This should be one of "AAStringSet", "DNAStringSet", or "RNAStringSet".
add2tbl	Logical or a character string specifying the table name in which to add the result.
batchSize	Integer specifying the number of sequences to process at a time.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

IdLengths is designed to efficiently determine the number of standard and non-standard characters in every sequence within a database. Standard and non-standard characters are defined with respect to the type of the sequences. For DNA and RNA sequences there are four standard characters and 11 non-standard characters (i.e., ambiguity codes). For amino acid sequences there are 20 standard and seven non-standard characters (including stops). Gap ("-"), missing ("."), and mask ("+") characters count toward the width but not the number of standard or non-standard characters.

Value

A data.frame with the number of standard characters, nonstandard characters, and width of each sequence. The row.names of the data.frame correspond to the "row_names" in the tblName of the dbFile.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

See Also

Add2DB

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 l <- IdLengths(db)
 head(l)
}
```

IdTaxa *Assign Sequences a Taxonomic Classification*

Description

Classifies sequences according to a training set by assigning a confidence to taxonomic labels for each taxonomic level.

Usage

```
IdTaxa(test,
       trainingSet,
       type = "extended",
       strand = "both",
       threshold = 60,
       bootstraps = 100,
       samples = L^0.47,
       minDescend = 0.98,
       fullLength = 0,
       processors = 1,
       verbose = TRUE)
```

Arguments

test	An AAStringSet, DNAStringSet, or RNAStringSet of unaligned sequences.
trainingSet	An object of class Taxa and subclass Train compatible with the class of test.
type	Character string indicating the type of output desired. This should be one of "extended" or "collapsed". (See value section below.)
strand	Character string indicating the orientation of the test sequences relative to the trainingSet. This should be one of "both", "top", or "bottom". The top strand is defined as the input test sequences being in the same orientation as the trainingSet, and the bottom strand is its reverse complement orientation. The default of "both" will classify using both orientations and choose the result with highest confidence. Choosing the correct strand will make classification over 2-fold faster, assuming that all of the reads are in the same orientation. Note that strand is ignored when test is an AAStringSet.
threshold	Numeric specifying the confidence at which to truncate the output taxonomic classifications. Lower values of threshold will classify deeper into the taxonomic tree at the expense of accuracy, and vice versa for higher values of threshold.
bootstraps	Integer giving the maximum number of bootstrap replicates to perform for each sequence. The number of bootstrap replicates is set automatically such that (on average) 99% of k-mers are sampled in each test sequence, unless the maximum bootstraps is reached.
samples	A function or call written as a function of 'L', which will evaluate to a numeric vector the same length as 'L'. Typically of the form "A + B*L^C", where 'A', 'B', and 'C' are constants.
minDescend	Numeric giving the minimum fraction of bootstraps required to descend the tree during the initial tree descend phase of the algorithm. Higher values are less likely to descend the tree, causing direct comparison against more sequences in the trainingSet. Lower values may increase classification speed at the expense of accuracy. Suggested values are between 1.0 and 0.9.
fullLength	Numeric specifying the fold-difference in sequence lengths between sequences in test and trainingSet that is allowable, or 0 (the default) to consider all sequences in trainingSet regardless of length. Can be specified as either a single numeric (> 1), or two numerics specifying the upper and lower fold-difference. If fullLength is between 0 and 1 (exclusive), the fold-difference is inferred from the length variability among sequences belonging to each class based on the foldDifference quantiles. For example, setting fullLength to 0.99 would use the 1st and 99th percentile of intra-group length variability from the trainingSet. In the case of full-length sequences, specifying fullLength can improve both speed and accuracy by using sequence length as a pre-filter to classification. Note that fullLength should only be greater than 0 when both the test and trainingSet consist of full-length sequences.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

Sequences in test are each assigned a taxonomic classification based on the trainingSet created with LearnTaxa. Each taxonomic level is given a confidence between 0% and 100%, and the taxonomy is truncated where confidence drops below threshold. If the taxonomic classification was truncated, the last group is labeled with "unclassified_" followed by the final taxon's name. Note that the reported confidence is not a p-value but does directly relate to a given classification's probability of being wrong. The default threshold of 60% is intended to minimize the rate of incorrect classifications. Lower values of threshold (e.g., 50%) may be preferred to increase the taxonomic depth of classifications. Values of 60% or 50% are recommended for nucleotide sequences and 50% or 40% for amino acid sequences.

Value

If type is "extended" (the default) then an object of class Taxa and subclass Train is returned. This is stored as a list with elements corresponding to their respective sequence in test. Each list element contains components:

taxon	A character vector containing the taxa to which the sequence was assigned.
confidence	A numeric vector giving the corresponding percent confidence for each taxon.
rank	If the classifier was trained with a set of ranks, a character vector containing the rank name of each taxon.

If type is "collapsed" then a character vector is returned with the taxonomic assignment for each sequence. This takes the repeating form "Taxon name [rank, confidence%]; ..." if ranks were supplied during training, or "Taxon name [confidence%]; ..." otherwise.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Murali, A., et al. (2018). IDTAXA: a novel approach for accurate taxonomic classification of microbiome sequences. Microbiome, 6, 140. https://doi.org/10.1186/s40168-018-0521-5

Cooley, N. and Wright, E. (2021). Accurate annotation of protein coding sequences with IDTAXA. NAR Genomics and Bioinformatics, **3(3)**. https://doi.org/10.1093/nargab/lqab080

See Also

LearnTaxa, Taxa-class

Run vignette("ClassifySequences", package = "DECIPHER") to see a related vignette.

Examples

```
data("TrainingSet_16S")

# import test sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
```

```
# remove any gaps in the sequences
dna <- RemoveGaps(dna)

# classify the test sequences
ids <- IdTaxa(dna, TrainingSet_16S, strand="top")
ids

# view the results
plot(ids, TrainingSet_16S)
```

IndexSeqs *Build an inverted index*

Description

Builds an inverted index from a set of amino acid, DNA, or RNA sequences.

Usage

```
IndexSeqs(subject,
          K,
          sensitivity,
          percentIdentity,
          patternLength,
          step = 1,
          alphabet = AA_REDUCED[[171]],
          maskRepeats = TRUE,
          maskLCRs = TRUE,
          maskNumerous = TRUE,
          batchSize = 1e+07,
          processors = 1,
          verbose = TRUE)
```

Arguments

subject	An AAStringSet, DNAStringSet, or RNAStringSet object of target (unaligned) sequences to use as the subject of the search.
K	Integer providing the k-mer length. Typical values are 5 to 7 for amino acids (in the default alphabet) and 8 to 12 for nucleotides.
sensitivity	Numeric giving the goal search sensitivity, which is used in the absence of specifying K. Typically near, but always less than, 1.
percentIdentity	Numeric identifying the goal percent identity for the given sensitivity, which is used in the absence of specifying K. Values closer to 100 allow for larger K and, thereby, faster searches.

patternLength	Integer setting the expected (minimum) length of query sequences, which is used in the absence of specifying K.
step	Integer determining the number of positions between the start of adjacent k-mers. Must be between 1 and K. Larger values reduce the memory required for the inverted index at the expense of search sensitivity.
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between protein sequences. A non-reduced amino acid alphabet can be used by setting alphabet equal to AA_STANDARD. Note that choice of alphabet affects the optimal value of K, with smaller alphabets requiring larger K and vice versa.
maskRepeats	Logical specifying whether to mask repeats in the subject. (See details section below.)
maskLCRs	Logical indicating whether to mask low complexity regions in the subject. (See details section below.)
maskNumerous	Logical indicating whether to mask frequent k-mers in the subject, or a positive numeric specifying the degree of masking to apply. (See details section below.)
batchSize	Integer defining the number of sequences to process in a batch. Smaller values reduce the function's memory footprint, potentially at the cost of increased runtime.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

An InvertedIndex object functions much like a the index at the back of a book, except with the goal of finding homologous sequence regions. It primarily contains the locations of length K subsequences (i.e., k-mers) in subject. Only the set of unmasked k-mers separated every step positions are stored. In general, lower values of K and, especially, step are preferable for improving search sensitivity and specificity. If an appropriate value for K is unknown, it is possible to automatically calculate K by providing a goal search sensitivity for sequences of patternLength positions with a given percentIdentity to a target sequence. The fastest matching value of K will be automatically selected, or an error will be returned when the constraints cannot be met.

Masking repeats and low complexity regions helps to avoid spurious hits that are not related to sequence homology (i.e., common descent), while masking extremely frequent k-mers can improve search speed and decrease the size of the inverted index. If maskRepeats is TRUE (the default), masking is first applied to tandem repeats, while maintaining the first copy of the repeat. Next, if maskLCRs is TRUE (the default), any remaining low complexity regions are masked that are at least as long as the k-mer length (i.e., K). Finally, if maskNumerous is TRUE (the default), extremely frequent k-mers that remain are masked on a per sequence basis. It is also possible to control the degree of masking frequent k-mers by supplying a positive number for maskNumerous, representing the -log(p-value) of observing that many k-mers by chance. Giving larger positive values for maskNumerous will mask fewer k-mers.

Proper masking is critical to balance search speed and sensitivity. When mapping reads to a subject genome it is often best to leave maskNumerous as TRUE but set maskRepeats and maskLCRs

to FALSE. This is because masking whole regions may preclude some reads from mapping, whereas masking the most frequent k-mers is sometimes necessary for speed. In contrast, when the subject is a set of shorter sequences (e.g., proteins) there are rarely extremely frequent k-mers in a single sequence, so maskNumerous has less (or no) effect. In this case, keeping maskRepeats and maskLCRs as TRUE will prevent k-mer matches between sequences that are not related by common descent. This is because some biological mechanisms can generate similar sequence patterns in unrelated sequences, which would otherwise mislead the statistical model of homology based on the likelihood of shared k-mers.

Value

An object of class InvertedIndex, which is stored as a list with components:

k	The input or automatically calculated value for K.
step	The input value of step.
alphabet	Numbered letter groups present in the sequence alphabet.
frequency	Numeric frequencies of each letter grouping.
count	The number of times every possible (unmasked) k-mer was observed.
length	An integer giving the number of (unmasked) k-mers in each sequence in subject.
location	The location of each k-mer within every sequence in subject, ordered by k-mer.
index	The index of each k-mer within every sequence in subject, ordered by k-mer.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2024) "Fast and Flexible Search for Homologous Biological Sequences with DECIPHER v3". The R Journal, **16(2)**, 191-200.

See Also

SearchIndex, AlignPairs

Run vignette("SearchForResearch", package = "DECIPHER") to see a related vignette.

Examples

```
# import target sequences
fas <- system.file("extdata", "PlanctobacteriaNamedGenes.fas.gz", package="DECIPHER")
seqs <- readAAStringSet(fas)

# build an inverted index, specifying K
index <- IndexSeqs(seqs, K=6L)
index # K = 6

# alternatively, determine K automatically
index <- IndexSeqs(seqs, sensitivity=0.99, percentIdentity=60, patternLength=300)
index # K = 3
```

InferDemography *Infer Demographic History from Allele Frequencies*

Description

Fits a population genetics model to an input site frequency spectrum (SFS) or one derived from aligned sequences. Returns estimated effective population sizes at time intervals in the past.

Usage

```
InferDemography(x,
                readingFrame = NA,
                mu = 1e-08,
                ploidy = 1,
                informationCriterion = "BIC",
                showPlot = FALSE,
                verbose = TRUE)
```

Arguments

x	A numeric vector of folded site frequencies (i.e., SFS[0], SFS[1], SFS[2], ..., SFS[n/2]), or a DNAStringSet or RNAStringSet of n aligned sequences from which to derive the folded SFS.
readingFrame	Either NA to use all sites, or the readingFrame (i.e., 1, 2, or 3) to limit analysis to only third codon positions. Only applicable if x is a XStringSet.
mu	A numeric giving the mutation rate per site per generation, which linearly affects the scaling of inferred demographic variables.
ploidy	An integer providing the ploidy (e.g., 1 for haploid or 2 for diploid organisms).
informationCriterion	Character string specifying which information criterion (i.e., "AIC" or "BIC") to use in determining the number of time intervals, or an integer specifying the number of intervals in the output.
showPlot	Logical specifying whether to show the fitted site frequency spectrum and inferred changes in effective population size (i.e., stairway plot).
verbose	Logical indicating whether to display progress.

Details

The site frequency spectrum (SFS) represents the distribution of allele frequencies within a population, which is sensitive to natural selection and changes in population size (Johri, et al., 2022). Using sites that are presumed to be neutrally evolving, it is possible to reconstruct historical changes in population size. This approach assumes frequent recombination, migration, and an unstructured population. Effective population sizes are inferred as a step function corresponding to different levels of the coalescent, with transitions times scaled to generations in the past.

InferDemography implements an extensive, but inexhaustive, procedure for optimizing transition points based on the likelihood equations described in Lynch, et al. (2020). The results provide an estimate of population expansions and contractions that reasonably fit simulations of the Wright-Fisher model given sufficient data. Axis scaling is dependent on the mutation rate (mu) and ploidy, which affect the quantitative (but not qualitative) results. Error bounds can be determined through bootstrapping by repeated sampling of sequences in x with replacement or by Poisson sampling frequency classes in x.

Increasing the number of sampled input species and neutral sites permits more accurate inference, although all sequences must originate from the same population. For this reason, the observed SFS that is output can be aggregated across multiple gene alignments of the same organisms and then used as input to increase the total number of polymorphisms. Notably, some organisms' life histories may poorly fit the standard Kingman coalescent used here (Freund, et al., 2023), and it is important to always verify a close match between the observed SFS and estimated SFS.

This approach requires multiple sequences sampled from a population with minimal divergence, since sites with more than two different alleles are discarded under the infinite sites model. The results can be used to compare demographics of genes within a species or contrast different populations' histories. In general, the trajectory of population expansions and contractions can provide insights into the balance between drift and selective forces previously acting on a population or gene.

Value

A named numeric vector with the following elements: (1) Intervals - number of different effective population sizes
(2) LogLikelihood - fitted model's log-likelihood
(3) Time - estimated time boundaries of each interval in units of generations ago
(4) Ne - estimated effective population size during each interval
(5) Observed - the observed (or input) folded site frequency spectrum
(6) Estimated - the estimated (fitted) folded site frequency spectrum

Different Time and Ne estimates are named by their merger level in the coalescent (n to 2).

Note

It is possible to exclude specific frequency classes in x by specifying them as NA. For example, if singletons (i.e., SFS[1]) are deemed unreliable due to sequencing error, it is feasible to exclude them by setting x[2] to NA, where x is the input numeric vector of folded site frequencies.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Freund, F., et al. (2023). Interpreting the pervasive observation of U-shaped Site Frequency Spectra. PLOS Genetics, **19(3)**, e1010677.

Johri, P., et al. (2022). Recommendations for improving statistical inference in population genomics. PLOS Biology, **20(5)**, e3001669.

Lynch, M., et al. (2020). Inference of Historical Population-Size Changes with Allele-Frequency Data. G3, **10(1)**, 211-223.

See Also

InferRecombination, InferSelection

Run `vignette("PopulationGenetics", package = "DECIPHER")` to see a related vignette.

Examples

```
# example of providing a folded SFS from the supplement of Lynch, et al. (2020)
SFS <- c(9526, 3998, 2315, 1487, 1075, 873, 689, 570, 567, 627, 547, 605, 460, 573, 236)
SFS <- c(1e7 - sum(SFS), SFS) # add monomorphic sites as first value
InferDemography(SFS, mu=1.2e-8, ploidy=2, show=TRUE)

# resample the folded SFS to test for stability
SFS2 <- rbinom(length(SFS), sum(SFS), SFS/sum(SFS))
InferDemography(SFS2, mu=1.2e-8, ploidy=2, show=TRUE)

# example of providing sequences from a (haploid) species
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna <- dna[startsWith(names(dna), "Helicobacter pylori")]
DNA <- AlignTranslation(dna) # align the translation then reverse translate
DNA
InferDemography(DNA, readingFrame=1, mu=1e-9, ploidy=1, show=TRUE)
```

InferRecombination *Infer Recombination Parameters from Correlation Profiles*

Description

Derives a correlation profile from aligned sequences and fits a population genetics model of recombination. Returns recombination parameters along with the fitted correlation profile.

Usage

```
InferRecombination(x,
                   readingFrame = NA,
                   position = 1:3,
                   N = 150,
                   showPlot = FALSE,
                   verbose = TRUE)
```

Arguments

x	A DNAStringSet or RNAStringSet of aligned sequences, or a list or XStringSetList containing multiple XStringSet objects.
readingFrame	Either NA, a numeric vector giving a single reading frame for all alignments, or an individual reading frame for each alignment in x. The readingFrame can be either 1, 2, 3 to signify codons beginning on the first, second, and third nucleotide position, respectively. If NA (the default), analysis is performed without reference to a reading frame. (See details section below.)
position	Numeric vector containing the codon position(s) to be analyzed, or NA to analyze all positions together. Typically, the third (3) codon position is analyzed for coding sequences and all positions (NA) for non-coding sequences. (See details section below.)
N	Numeric giving the maximum number of positions away from the reference (initial) position to include in the correlation profile.
showPlot	Logical specifying whether or not to plot the correlation profile(s) and fitted curves.
verbose	Logical indicating whether to display progress.

Details

Recombination accelerates the adaptive evolution of many organisms. The transfer of genetic fragments leaves behind a signature in the form of decaying autocorrelation among substitutions. This signal can be visualized with a correlation profile that shows the probability of a genetic difference at increasing distances away from another genetic difference. Flat correlation profiles are evidence for an absence of recombination since the last common ancestor. In contrast, more "L" shaped correlation profiles result from greater degrees of recombination (Steinberg, et al., 2023).

InferRecombination uses maximum likelihood estimation to fit a coalescent-based model to a correlation profile derived from one or more sets of aligned sequences (x). The model assumes the sequences correspond to a population whose genealogical structure is characterized by a single average coalescence rate. This assumption is automatically true for a pair of sequences and may hold true within (e.g., intra-species) or between clusters of sequences (Steinberg, et al., 2022). The model can be applied to any organisms belonging to a focal population recombining with genetic fragments from an external pool.

The parametric model consist of three free variables that are fitted to the measured correlation profile (Lin & Kussell, 2019). These variables can be used to infer six other parameters of interest corresponding to the sample (x) or (external) pool in which they recombine. (See value section below.) This approach is fast, accurate, and does not rely on phylogenetic reconstruction. Variability can be quantified by comparing results across alignments and confidence intervals can be determined by bootstrapping sequences in the input alignments (x).

It is possible to indicate the readingFrame and codon position(s) that should be assessed. Typically, the analysis is performed on the third position of synonymous codons to mitigate the influence of adaptive substitutions. If the readingFrame is NA, positions are analyze with respect to the (initial) substitutions, and a positional pattern may emerge due to variability in substitution rates at each codon position. It is also possible to include all positions (readingFrame=NA), such as when analyzing non-coding sequences.

The output consists of a matrix of measured values and inferred parameters. Of particular note are the coverage, ratio, and theta_pool. The recombined coverage is 0% when the sequences have evolved independently since their last common ancestor and 100% when the (input) sample has recombined all of its nucleotides with the (external) pool. The ratio of recombination rate to mutation rate can be interpreted as the ratio of substitutions due to recombination relative to point mutation. This ratio is greater than 1 when recombination contributes more to genetic diversity than point mutation, and vice versa when less than 1. The value of theta_pool quantifies the diversity of the external genetic pools accessible by the sampled sequences through recombination. It can be used as a proxy for the pairwise separation between genomes, since a low value of theta_pool indicates the genomes are intermixing with the same pool (Lin & Kussell, 2019).

Note that a sufficient number of mutations is required for curve fitting, which can become problematic when d_sample is close to zero. A good rule of thumb is that the number of observed mutations per profile position, which can be approximated by "Profile 1" multiplied by "Count 1", should be above about 1000 for dependable curve fitting. (See examples section below.)

Value

A matrix with named rows for each value and a column for each position. The meaning of each parameter is described in Lin & Kussell (2019).

The first three rows are the fitted parameters:
(1) fragment - mean recombined genetic fragment size (in base pairs)
(2) theta_sample - mutational divergence within the (input) sample
(3) phi_sample - recombinational divergence within the (input) sample

The next six rows are derived from the fitted parameters:
(4) theta_pool - mutational divergence within the (external) pool
(5)phi_pool - recombinational divergence within the (external) pool
(6) ratio - the relative rate of recombination to mutation (also known as r/m)
(7) coverage - proportion of sites in the (input) sample whose diversity originated from the (external) pool
(8) d_pool - pairwise diversity of the (external) pool (i.e., probability that any two sequences in the pool differ at a site)
(9) d_clonal - pairwise diversity of the (input) sample due to accumulated substitutions (i.e., probability any two sequences in the sample differ at a site due to clonal mutation)

The remaining rows contain fixed values:
(10) d_sample - measured pairwise diversity due to both recombination and mutation (i.e., probability two sequences differ at a site)
(11) Position - distance away from the initial substitution (in base pairs)
(12) Profile - measured probability of a difference at each position (i.e., the measured correlation profile)
(13) Fitted - estimated probability of a difference at each position (i.e., the fitted correlation profile)
(14) Count - Total number of mutations analyzed at each position (i.e., the normalization factor of the correlation profile)

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Lin, M. & Kussell, E. (2019). Inferring bacterial recombination rates from large-scale sequencing datasets. Nature Methods, **16(2)**, 199-204.

Steinberg, A., et al. (2022). Core genes can have higher recombination rates than accessory genes within global microbial populations. eLife, **11**, e78533.

Steinberg, A., et al. (2023). Correlated substitutions reveal SARS-like coronaviruses recombine frequently with a diverse set of structured gene pools. PNAS, **120(5)**, e2206945119.

See Also

InferDemography, InferSelection

Run `vignette("PopulationGenetics", package = "DECIPHER")` to see a related vignette.

Examples

```
# example for an alignment of coding sequences
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
DNA <- readDNAStringSet(fas)
DNA <- DNA[startsWith(names(DNA), "Helicobacter")] # subset to species
DNA <- AlignTranslation(DNA)
ans <- InferRecombination(DNA, position=3, readingFrame=1, showPlot=TRUE)
head(ans, n=10)

# example for an alignment of non-coding sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
DNA <- readDNAStringSet(fas)
ans <- InferRecombination(DNA, position=NA, showPlot=TRUE)
head(ans, n=10)
ans["Profile 1",]*ans["Count 1",] # ideally > 1,000 for stable fitting

if (require("RSQLite", quietly=TRUE)) {
 # example of inferring recombination among genomes
 db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
 synteny <- FindSynteny(db, minScore=50)
 DNA <- AlignSynteny(synteny, db)
 ans <- lapply(DNA, InferRecombination, position=3, verbose=FALSE)
 ans <- setNames(do.call(cbind, ans), names(DNA))
 pairs(t(ans[1:10,]))
 DNA <- do.call(c, unname(DNA)) # combine all alignments
 ans <- InferRecombination(DNA, showPlot=TRUE)
 head(ans, n=10)
}
```

`InferSelection` *Infer Codon Selection on Protein Coding Sequences*

Description

Infers the magnitude and direction of natural selection by fitting a population genetics model to aligned protein coding sequences. Returns estimated selection parameters, including the Ka/Ks ratio (omega) by region.

Usage

```
InferSelection(myDNAStringSet,
               readingFrame = 1L,
               windowSize = NA,
               tolerance = 5e-05,
               geneticCode = GENETIC_CODE,
               showPlot = FALSE,
               verbose = TRUE)
```

Arguments

myDNAStringSet	A DNAStringSet of aligned sequences.
readingFrame	A numeric vector giving the starting position of the first codon in the alignment, either 1 (the default), 2, or 3.
windowSize	Either NA to estimate the average Ka/Ks ratio (omega) across all codons, or an integer giving the size (in codons) of non-overlapping windows in which to estimate individual Ka/Ks ratios (omegas). For example, setting the windowSize to 1 will estimate an omega for every codon.
tolerance	Numeric determining the relative convergence tolerance. Optimization will cease when the relative likelihood has changed by less than tolerance.
geneticCode	A character vector giving the genetic code in the same format as GENETIC_CODE (the default).
showPlot	Logical specifying whether to show the estimated Ka/Ks ratio(s) (omega) along the alignment. More significant p-values are displayed in a brighter green color when showPlot is TRUE.
verbose	Logical indicating whether to display progress.

Details

The Ka/Ks ratio, also known as omega or dN/dS, is a measure of the magnitude and direction of natural selection operating on protein coding genes. It represents the rate of non-synonymous versus synonymous codon substitutions, and a bias in this rate is indicative of selection. Ratios of 1 are expected under neutral evolution, but ratios less than 1 are typically observed due to negative (purifying) selection acting to maintain the protein sequence. In contrast, ratios greater than 1 are of particular interest, as they may represent the presence of positive (Darwinian) selection for protein changes. Ratios are rarely greater than 1 on average for entire coding sequences, so it is common to rank genes by the fraction (or total number) of codons under positive selection (Moutinho, et al., 2023).

InferSelection fits a three parameter NY98 substitution rate matrix (Nielsen & Yang, 1998) to the observed distribution of codons at sites in an alignment using a population genetics method

(Wilson, D & CRyPTIC Consortium, 2020). This approach derives estimates of kappa (transition/transversion ratio), theta (population scaled mutation rate), and omega (Ka/Ks ratio). The Ka/Ks ratio (omega) can be estimated for the entire alignment or non-overlapping windows of `windowSize` codons. This estimation approach effectively considers omega as a mutational bias in the rate of non-synonymous versus synonymous changes. Codon frequencies are derived from nucleotide frequencies in the input sequences and fitted parameters are determined by maximum likelihood estimation.

The method used by `InferSelection` requires a low mutation rate approximation, so it is expected to work best on a single population from the same species. It is noteworthy that omega is known to be closer to 1 than expected when measured within populations, rather than between populations where mutations are fixed (Kryazhimskiy & Plotkin, 2008). This prevents omega from being accurately transformed into (and interpreted as) a population-scaled selection coefficient. Notwithstanding this limitation, the advantage of this approximation is that a phylogenetic tree is not required, and the method can easily scale to large numbers of input sequences. Many sequences are typically needed for accurate estimates since variation is relatively rare within populations.

The population genetics model employed here also assumes independence between sites. Nevertheless, simulations show the method is robust to model violations when there is a lack of recombination (Wilson, D & CRyPTIC Consortium, 2020). The independence assumption confers the advantages of scalability, simple handling of missing data (e.g., gaps), and no requirement for haplotype information. Sequences are assumed to be randomly sampled from an unstructured population with constant population size over time. The population sample is very important, as biased sampling could result in correspondingly biased inferences about selection.

Statistical significance is obtained from a likelihood ratio test based on the chi-squared distribution with 1 degree of freedom (Anisimova, et al., 2001), comparing each fitted value of omega to 1 (i.e., neutrality). Simulations of the coalescent process suggest that the number of sequences in `myXStringSet` should be at least `200/windowSize` for reasonable power to detect positive selection. That is, about 200 sequences are required to identify a considerable fraction of codon-level selection (i.e., when `windowSize` is 1), but only a couple of sequences are required to observe overall selection on typical length coding sequences (>= 100 codons when `windowSize` is `NA`). Alternative estimates of statistical significance can be obtained by bootstrapping the input sequences when their number is sufficiently large.

Value

A named numeric vector with the following elements: (1) `LogLikelihood` - fitted model's log-likelihood
(2) `theta` - expected substitution rate in units of 2*ploidy*Ne generations
(3) `kappa` - estimated transition to transversion ratio
(4) `omega` - estimated Ka/Ks (dN/dS) ratio(s) per window
(5) `pvalue` - corresponding p-value with null hypothesis omega = 1

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Anisimova, M., et al. (2001). Accuracy and power of the likelihood ratio test in detecting adaptive molecular evolution. Molecular Biology and Evolution, **18(8)**, 1585-1592.

Kryazhimskiy, S. & Plotkin, J. (2008). The population genetics of dN/dS. PLoS Genetics, **4(12)**, e1000304.

Moutinho, A., et al. (2019). Variation of the adaptive substitution rate between species and within genomes. Evolutionary Ecology, **34(3)**, 315-338.

Nielsen, R. & Yang, Z. (1998). Likelihood models for detecting positively selected amino acid sites and applications to the HIV-1 envelope gene. Genetics, **148(3)**, 929-936.

Wilson, D. & CRyPTIC Consortium. (2020) GenomegaMap: Within-Species Genome-Wide dN/dS Estimation from over 10,000 Genomes. Molecular Biology and Evolution, **37(8)**, 2450-2460.

See Also

InferDemography, InferRecombination

Run `vignette("PopulationGenetics", package = "DECIPHER")` to see a related vignette.

Examples

```
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
DNA <- readDNAStringSet(fas)
DNA <- DNA[startsWith(names(DNA), "Helicobacter")] # subset to species
DNA <- AlignTranslation(DNA)

InferSelection(DNA, windowSize=NA, showPlot=TRUE)
# note: set windowSize=1 to estimate omega per codon
```

InvertedIndex *InvertedIndex objects*

Description

`InvertedIndex` objects store k-mer locations and indexes in a set of sequences.

Usage

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

Arguments

`x`	An object of class `InvertedIndex`.
`...`	Other optional parameters.

Details

Objects of class `InvertedIndex` are stored as a list. The function `IndexSeqs` returns an object of class `InvertedIndex`. The information stored in an `InvertedIndex` can be displayed with `print`.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2024) "Fast and Flexible Search for Homologous Biological Sequences with DECIPHER v3". The R Journal, **16(2)**, 191-200.

See Also

`IndexSeqs`, `SearchIndex`

Examples

```
# import target sequences
fas <- system.file("extdata", "PlanctobacteriaNamedGenes.fas.gz", package="DECIPHER")
seqs <- readAAStringSet(fas)

# build an inverted index
index <- IndexSeqs(seqs, K=6L)
index # print the index
```

LearnNonCoding *Learn a Non-Coding RNA Model*

Description

Learns a compact representation of patterns representing a set of non-coding RNAs belonging to the same family.

Usage

```
LearnNonCoding(myXStringSet,
               threshold = 0.3,
               weight = NA,
               maxLoopLength = 500,
               maxPatterns = 20,
               scoreDependence = FALSE,
               structure = NULL,
               processors = 1)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet object of aligned sequence representatives belonging to the same non-coding RNA family.
threshold	Numeric specifying the minimum relative frequency of patterns to consider during learning.
weight	Either a numeric vector of weights for each sequence, a single number implying equal weights, or NA (the default) to automatically calculate sequence weights based on myXStringSet.
maxLoopLength	Numeric giving the maximum length of conserved hairpin loops to consider.
maxPatterns	A numeric vector of length two specifying the maximum number of motifs and hairpins, respectively, or a single numeric giving the maximum for each.
scoreDependence	Logical determining whether to record a log-odds score for dependencies between patterns. The default (FALSE) is recommended for most non-coding RNA families.
structure	Either a character string providing the consensus secondary structure in dot bracket notation, a matrix of paired positions in the first two columns, or NULL (the default) to predict the consensus secondary structure with PredictDBN.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.

Details

Non-coding RNAs belonging to the same family typically have conserved sequence motifs, secondary structure elements, and k-mer frequencies that can be used to identify members of the family. LearnNonCoding identifies these conserved patterns and determines which are best for identifying the non-coding RNA relative to a random sequence background. Sequence motifs and hairpins are defined relative to their distance from the start or end of the non-coding RNA, allowing the precise and rapid identification of the boundaries of any matches to the non-coding RNA in a genome.

Value

An object of class NonCoding.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2021). FindNonCoding: rapid and simple detection of non-coding RNAs in genomes. Bioinformatics. https://doi.org/10.1093/bioinformatics/btab708

See Also

FindNonCoding, NonCoding-class

Run vignette("FindingNonCodingRNAs", package = "DECIPHER") to see a related vignette.

Examples

```
# import a family of non-coding RNAs
fas_path <- system.file("extdata",
 "IhtA.fas",
 package="DECIPHER")
rna <- readRNAStringSet(fas_path)
rna

# align the sequences
RNA <- AlignSeqs(rna,
 useStructures=FALSE) # sacrifice accuracy for speed in example
RNA

y <- LearnNonCoding(RNA)
y
y[["motifs"]]
y[["hairpins"]]
head(y[["kmers"]])
```

LearnTaxa *Train a Classifier for Assigning Taxonomy*

Description

Trains a classifier based on a reference taxonomy containing sequence representatives assigned to taxonomic groups.

Usage

```
LearnTaxa(train,
          taxonomy,
          rank = NULL,
          K = NULL,
          N = 500,
          minFraction = 0.01,
          maxFraction = 0.06,
          maxIterations = 10,
          multiplier = 100,
          maxChildren = 200,
          alphabet = AA_REDUCED[[62]],
          verbose = TRUE)
```

Arguments

train	An AAStringSet, DNAStringSet, or RNAStringSet of unaligned sequences.
taxonomy	Character vector providing the reference taxonomic assignment for each sequence in train. Taxonomic ranks are separated by semicolons (";") beginning with "Root".

rank	Optionally, a data.frame with 5 named columns giving the "Index" (i.e., 0 to the number of unique taxa), "Name" (i.e., taxon name), "Parent" (i.e., "Index" of the parent taxon), "Level" (i.e., integer rank level), and "Rank" (e.g., "genus") of each taxonomic rank. This information is often provided in a separate "taxid" file along with publicly available training sequence sets.
K	Integer specifying the k-mer size or NULL (the default) to calculate the k-mer size automatically. The default value of K is such that matches between sequences are found by chance every N k-mers.
N	Numeric indicating the approximate number of k-mers that can be randomly selected before one is found by chance on average. For example, the default value of 500 will set K (when K is unspecified) so that every 500th k-mer is expected to match by chance.
minFraction	Numeric giving the minimum fraction of k-mers to sample during the initial tree descent phase of the classification algorithm. (See details section below.)
maxFraction	Numeric giving the maximum fraction of k-mers to sample during the initial tree descent phase of the classification algorithm. (See details section below.)
maxIterations	Integer specifying the maximum number of iterations to attempt re-classification of a training sequence before declaring it a "problem sequence". (See details section below.)
multiplier	Numeric indicating the degree to which individual sequences have control over the fraction of k-mers sampled at any edge during the initial tree descent phase of the classification algorithm. (See details section below.)
maxChildren	Integer giving the maximum number of child taxa of any taxon at which to consider further descending the taxonomic tree. A value of 1 will prevent use of the tree descent algorithm altogether. Lower values may decrease classification speed, but result in output objects that require less memory.
alphabet	Character vector of amino acid groupings used to reduce the 20 standard amino acids into smaller groups. Alphabet reduction helps to find more distant homologies between sequences. A non-reduced amino acid alphabet can be used by setting alphabet equal to AA_STANDARD.
verbose	Logical indicating whether to display progress.

Details

Learning about the training data is a two part process consisting of (i) forming a taxonomic tree and then (ii) ensuring that the training sequences can be correctly reclassified. The latter step relies on reclassifying the sequences in train by descending the taxonomic tree, a process termed "tree descent". Ultimately, the goal of tree descent is to quickly and accurately narrow the selection of groups where a sequence may belong. During the learning process, tree descent is tuned so that it performs well when classifying new sequences.

The process of training the classifier first involves learning the taxonomic tree spanning all of the reference sequences in train. Typically, reference taxonomic classifications are provided by an authoritative source, oftentimes along with a "taxid" file containing taxonomic rank information. The taxonomic tree may contain any number of levels (e.g., Root, Phylum, Class, Order, Family, Genus) as long as they are hierarchically nested and always begin with "Root".

The second phase of training the classifier, tree descent, involves learning the optimal set of k-mers for discerning between the different sub-groups under each edge. Here a fraction of the k-mers with the greatest discerning power are matched to a training sequence, and this process is repeated with 100 random subsamples to decide on the set of possible taxonomic groups to which a training sequence may belong.

The learning process works by attempting to correctly re-classify each training sequence in the taxonomy. Initially, `maxFraction` of informative k-mers are repeatedly sampled at each edge during tree descent. Training sequences that are incorrectly classified at an edge will lower the fraction of k-mers that are sampled by an amount that is proportional to `multiplier`. As the fraction of sampled k-mers decreases, the tree descent process terminates at higher rank levels.

A major advantage of tree descent is that it both speeds up the classification process and indicates where the training set likely contains mislabeled sequences or incorrectly-placed taxonomic groups. Training sequences that are not correctly classified within `maxIterations` are marked as "problem sequences", because it is likely that they are mislabeled. If enough sequences have difficulty being correctly classified at an edge that the fraction drops below `minFraction`, then the edge is recorded as a "problem group".

The final result is an object that can be used for classification with `IdTaxa`, as well as information about `train` that could be used to help correct any errors in the taxonomy.

Value

An object of class `Taxa` and subclass `Train`, which is stored as a list with components:

`taxonomy`	A character vector containing all possible groups in the taxonomy.
`taxa`	A character vector containing the basal taxon in each taxonomy.
`ranks`	A character vector of rank names for each taxon, or `NULL` if `rank` information was not supplied.
`levels`	Integer giving the rank level of each taxon.
`children`	A list containing the index of all children in the taxonomy for each taxon.
`parents`	An integer providing the index of the parent for each taxon.
`fraction`	A numeric between `minFraction` and `maxFraction` that represents the learned fraction of informative k-mers to sample for each taxon during the initial tree descent phase of the classification algorithm. Problem groups are marked by a fraction of `NA`.
`sequences`	List containing the integer indices of sequences in `train` belonging to each taxon.
`kmers`	List containing the unique sorted k-mers (converted to integers) belonging to each sequence in `train`.
`crossIndex`	Integer indicating the index in taxonomy of each sequence's taxonomic label.
`K`	The value of `K` provided as input.
`IDFweights`	Numeric vector of length `4^K` providing the inverse document frequency weight for each k-mer.
`decisionKmers`	List of informative k-mers and their associated relative frequencies for each internal edge in the taxonomy.

problemSequences
: A data.frame providing the "Index", "Expected" label, and "Predicted" taxon for sequences that could not be correctly classified during the initial tree descent phase of the algorithm.

problemGroups
: Character vector containing any taxonomic groups that repeatedly had problems with correctly re-classifying sequences in train during the initial tree descent phase of the classification algorithm. Problem groups likely indicate that a number of the sequences (or an entire group of sequences) assigned to the problem group are incorrectly placed in the taxonomic tree.

alphabet
: The alphabet when train is an "AAStringSet", otherwise NULL.

Note

If K is NULL, the automatically determined value of K might be too large for some computers, resulting in an error. In such cases it is recommended that K be manually set to a smaller value.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Murali, A., et al. (2018). IDTAXA: a novel approach for accurate taxonomic classification of microbiome sequences. Microbiome, 6, 140. https://doi.org/10.1186/s40168-018-0521-5

See Also

IdTaxa, Taxa-class

Run vignette("ClassifySequences", package = "DECIPHER") to see a related vignette.

Examples

```
# import training sequences
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

# parse the headers to obtain a taxonomy
s <- strsplit(names(dna), " ")
genus <- sapply(s, `[`, 1)
species <- sapply(s, `[`, 2)
taxonomy <- paste("Root", genus, species, sep="; ")
head(taxonomy)

# train the classifier
trainingSet <- LearnTaxa(dna, taxonomy)
trainingSet

# view information about the classifier
plot(trainingSet)
```

```
## Not run:
# train the classifier with amino acid sequences
aa <- translate(dna)
trainingSetAA <- LearnTaxa(aa, taxonomy)
trainingSetAA

## End(Not run)
```

MapCharacters *Map Changes in Ancestral Character States*

Description

Maps character changes on a phylogenetic tree containing reconstructed ancestral states.

Usage

```
MapCharacters(x,
              refPositions = seq_len(nchar(attr(x, "state")[1])),
              labelEdges = FALSE,
              type = "dendrogram",
              chars = LETTERS,
              ignoreIndels = TRUE)
```

Arguments

x	An object of class dendrogram with "state" attributes for each node.
refPositions	Numeric vector of reference positions in the original sequence alignment. Only changes at refPositions are reported, and state changes are labeled according to their position in refPositions.
labelEdges	Logical determining whether to label edges with the number of changes along each edge.
type	Character string indicating the type of output desired. This should be one of "dendrogram", "table", or "both". (See value section below.)
chars	Character vector specifying the characters to consider in state changes at each site. The default (LETTERS) is to consider any upper case letter. Alternatively, chars could be AA_STANDARD, DNA_BASES, or RNA_BASES.
ignoreIndels	Logical specifying whether to report insertions and deletions (indels). If TRUE (the default), only substitutions of one state with another are reported.

Details

Ancestral state reconstruction affords the ability to identify character changes that occurred along edges of a rooted phylogenetic tree. Character changes are reported according to their index in refPositions. If ignoreIndels is FALSE, adjacent insertions and deletions are merged into single changes occurring at their first position. The table of changes can be used to identify parallel, convergent, and divergent mutations.

Value

If type is "dendrogram" (the default) then the original dendrogram x is returned with the addition of "change" attributes on every edge except the root. If type is "table" then a sorted table of character changes is returned with the most frequent parallel changes at the beginning. If type is "both" then a list of length 2 is provided containing both the dendrogram and table.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

Treeline

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

tree <- Treeline(myXStringSet=dna, reconstruct=TRUE)

out <- MapCharacters(tree,
                     labelEdges=TRUE,
                     type="both",
                     chars=DNA_BASES)

# plot the tree with defaults
tree <- out[[1]]
plot(tree, horiz=TRUE) # edges show number of changes

# color edges by number of changes
maxC <- 200 # changes at maximum of color spectrum
colors <- colorRampPalette(c("black", "darkgreen", "green"))(maxC)
colorEdges <- function(x) {
   num <- attr(x, "edgetext") + 1
   if (length(num)==0)
       return(x)
   if (num > maxC)
       num <- maxC
   attr(x, "edgePar") <- list(col=colors[num])
   attr(x, "edgetext") <- NULL
   return(x)
}
colorfulTree <- dendrapply(tree, colorEdges)
plot(colorfulTree, horiz=TRUE, leaflab="none")

# look at parallel changes (X->Y)
parallel <- out[[2]]
head(parallel) # parallel changes

# look at convergent changes (*->Y)
convergent <- gsub(".*?([0-9]+.*)", "\\1", names(parallel))
```

```
convergent <- tapply(parallel, convergent, sum)
convergent <- sort(convergent, decreasing=TRUE)
head(convergent)

# look at divergent changes (X->*)
divergent <- gsub("(.*[0-9]+).*", "\\1", names(parallel))
divergent <- tapply(parallel, divergent, sum)
divergent <- sort(divergent, decreasing=TRUE)
head(divergent)

# plot number of changes by position
changes <- gsub(".*?([0-9]+).*", "\\1", names(parallel))
changes <- tapply(parallel, changes, sum)
plot(as.numeric(names(changes)),
     changes,
     xlab="Alignment position",
     ylab="Total independent changes")
# functionally important regions tend to accept fewer changes
o <- order(as.numeric(names(changes)))
lines(as.numeric(names(changes))[o],
 filter(changes, rep(1/10, 10))[o],
 lwd=2, col="red")
legend("topleft", "Moving average", lwd=2, col="red")

# count cases of potential compensatory mutations
compensatory <- dendrapply(tree,
    function(x) {
        change <- attr(x, "change")
        pos <- as.numeric(gsub(".*?([0-9]+).*", "\\1", change))
        e <- expand.grid(seq_along(pos), seq_along(pos))
        e <- e[pos[e[, 1]] < pos[e[, 2]],]
        list(paste(change[e[, 1]], change[e[, 2]], sep=" & "))
    })
compensatory <- unlist(compensatory)
u <- unique(compensatory)
m <- match(compensatory, u)
m <- tabulate(m, length(u))
compensatory <- sort(setNames(m, u), decreasing=TRUE)
head(compensatory) # ranked list of concurrent mutations
```

MaskAlignment *Mask Highly Variable Regions of An Alignment*

Description

Automatically masks poorly aligned regions of an alignment based on sequence conservation and gap frequency.

Usage

```
MaskAlignment(myXStringSet,
              type = "sequences",
              windowSize = 4,
              threshold = 1,
              minFractionGaps = 0.2,
              maxFractionGaps = 1,
              includeTerminalGaps = FALSE,
              correction = FALSE,
              randomBackground = FALSE,
              showPlot = FALSE)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of aligned sequences.
type	Character string indicating the type of result desired. This should be one of "sequences", "ranges", or "values". (See value section below.)
windowSize	Integer value specifying the size of the region to the left and right of the center-point to use in calculating the moving average.
threshold	Numeric giving the average entropy in bits below which a region is masked.
minFractionGaps	Numeric specifying the minimum fraction of gaps in an alignment column to allow masking based on entropy. Values greater than zero prevent columns without gaps from being masked.
maxFractionGaps	Numeric specifying the maximum fraction of gaps in an alignment column to be masked.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("." or "-" characters on each end of the sequences) into the calculation of gap fraction.
correction	Logical indicating whether to apply a small-sample size correction to columns with few letters (Yu et al., 2015).
randomBackground	Logical determining whether background letter frequencies are determined directly from myXStringSet (the default) or a uniform distribution of letters.
showPlot	Logical specifying whether or not to show a plot of the positions that were kept or masked.

Details

Poorly aligned regions of a multiple sequence alignment may lead to incorrect results in downstream analyses. One method to mitigate their effects is to mask columns of the alignment that may be poorly aligned, such as highly-variable regions or regions with many insertions and deletions (gaps).

Highly variable regions are detected by their signature of having a low information content. Here, information content is defined by the relative entropy of a column in the alignment (Yu et al., 2015),

which is higher for conserved columns. The relative entropy is based on the background distribution of letter-frequencies in the alignment.

A moving average of `windowSize` nucleotides to the left and right of the center-point is applied to smooth noise in the information content signal along the sequence. Regions dropping below `threshold` bits or more than `maxFractionGaps` are masked.

Value

If `type` is `"sequences"` then an `XStringSet` object with masked ("+") positions. Otherwise, if `type` is `"ranges"` then an `IRanges` object giving the start and end positions of the masked columns. Else (`type` is `"values"`) a `data.frame` containing one row per site in the alignment and three columns of information:

`"entropy"`	The entropy score of each column, in units of bits.
`"gaps"`	For each column, the fraction of gap characters ("-" or ".").
`"mask"`	A logical vector indicating whether or not the column met the criteria for masking.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Yu, Y.-K., et al. (2015). Log-odds sequence logos. Bioinformatics, 31(3), 324-331. http://doi.org/10.1093/bioinformatics/btu

See Also

AlignSeqs, Treeline

Examples

```
fas <- system.file("extdata", "Streptomyces_ITS_aligned.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna # before masking
masked_dna <- MaskAlignment(dna, showPlot=TRUE)
masked_dna # after masking (adds "+" symbols)

# get the start and end ranges of masked columns
ranges <- MaskAlignment(dna, type="ranges")
ranges
replaceAt(dna, ranges) # remove the masked columns

# obtain the entropy scores of each column
values <- MaskAlignment(dna, type="values")
head(values)
```

MeltDNA *Simulate Melting of DNA*

Description

The denaturation of double-stranded DNA occurs over a range of temperatures. Beginning from a helical state, DNA will transition to a random-coil state as temperature is increased. MeltDNA predicts the positional helicity, melt curve, or its negative derivative at different temperatures.

Usage

```
MeltDNA(myDNAStringSet,
        type = "derivative",
        temps = 50:100,
        ions = 0.2)
```

Arguments

myDNAStringSet	A DNAStringSet object or character vector with one or more sequences in 5' to 3' orientation.
type	Character string indicating the type of results desired. This should be one of "derivative curves", "melt curves", or "positional probabilities".
temps	Numeric vector of temperatures (in degrees Celsius).
ions	Numeric giving the molar sodium equivalent ionic concentration. Values must be at least 0.01M.

Details

When designing a high resolution melt (HRM) assay, it is useful to be able to predict the results before performing the experiment. Multi-state models of DNA melting can provide near-qualitative agreement with experimental DNA melt curves obtained with quantitative PCR (qPCR). MeltDNA employs the algorithm of Tostesen et al. (2003) with an approximation for loop entropy that runs in nearly linear time and memory, which allows very long DNA sequences (up to 100,000 base pairs) to be analyzed.

Denaturation is a highly cooperative process whereby regions of double-stranded DNA tend to melt together. For short sequences (< 100 base pairs) there is typically a single transition from a helical to random-coil state. Longer sequences may exhibit more complex melting behavior with multiple peaks, as domains of the DNA melt at different temperatures. The melting curve represents the average fractional helicity (Theta) at each temperature, and can be used for genotyping with high resolution melt analysis.

Value

MeltDNA can return three types of results: positional helicity, melting curves, or the negative derivative of the melting curves. If type is "position", then a list is returned with one component for

each sequence in myDNAStringSet. Each list component contains a matrix with the probability of helicity (Theta) at each temperature (rows) and every position in the sequence (columns).

If type is "melt", then a matrix with the average Theta across the entire sequence is returned. This matrix has a row for each input temperature (temps), and a column for each sequence in myDNAStringSet. For example, the value in element [3, 4] is the average helicity of the fourth input sequence at the third input temperature. If type is "derivative" then the values in the matrix are the derivative of the melt curve at each temperature.

Note

MeltDNA uses nearest neighbor parameters from SantaLucia (1998).

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

SantaLucia, J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proceedings of the National Academy of Sciences, 95(4), 1460-1465.

Tostesen, E., et al. (2003). Speed-up of DNA melting algorithm with complete nearest neighbor properties. Biopolymers, 70(3), 364-376. doi:10.1002/bip.10495.

See Also

AmplifyDNA, CalculateEfficiencyPCR, DesignSignatures

Examples

```
fas <- system.file("extdata", "IDH2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

# plot the melt curve for the two alleles
temps <- seq(85, 100, 0.2)
m <- MeltDNA(dna,
             type="melt", temps=temps, ions=0.1)
matplot(temps, m,
        type="l", xlab="Temperature (\u00B0C)", ylab="Average Theta")
legend("topright", names(dna), lty=seq_along(dna), col=seq_along(dna))

# plot the negative derivative curve for a subsequence of the two alleles
temps <- seq(80, 95, 0.25)
m <- MeltDNA(subseq(dna, 492, 542),
             type="derivative", temps=temps)
matplot(temps, m,
        type="l", xlab="Temperature (\u00B0C)", ylab="-d(Theta)/dTemp")
legend("topright", names(dna), lty=seq_along(dna), col=seq_along(dna))

# plot the positional helicity profile for the IDH2 allele
temps <- seq(90.1, 90.5, 0.1)
m <- MeltDNA(dna[1],
```

```
            type="position", temps=temps, ions=0.1)
matplot(seq_len(dim(m[[1]])[2]), t(m[[1]]),
        type="l", xlab="Nucleotide Position", ylab="Theta")
temps <- formatC(temps, digits=1, format="f")
legend("topright", legend=paste(temps, "\u00B0C", sep=""),
        col=seq_along(temps), lty=seq_along(temps), bg="white")
```

MIQS *MIQS Amino Acid Substitution Matrix*

Description

The MIQS amino acid substitution matrix defined by Yamada & Tomii (2014).

Usage

```
data("MIQS")
```

Format

The format is: num [1:25, 1:25] 3.2 -1.3 -0.4 -0.4 1.5 -0.2 -0.4 0.4 -1.2 -1.3 ... - attr(*, "dimnames")=List of 2 ..$: chr [1:25] "A" "R" "N" "D" ... ..$: chr [1:25] "A" "R" "N" "D" ...

Details

Substitution matrix values represent the log-odds of observing an aligned pair of amino acids versus the likelihood of finding the pair by chance. Values in the MIQS matrix are in units of third-bits ($log(odds\ ratio) * 3/log(2)$).

Source

Yamada, K., & Tomii, K. (2014). Revisiting amino acid substitution matrices for identifying distantly related proteins. *Bioinformatics*, **30(3)**, 317-325.

Examples

```
data(MIQS)
MIQS["A", "R"] # score for A/R pairing

data(BLOSUM)
plot(BLOSUM[1:20, 1:20, "62"], MIQS[1:20, 1:20])
abline(a=0, b=1)
```

MMLSUM *MMLSUM Amino Acid Substitution Matrices*

Description

The MMLSUM amino acid substitution matrices defined by Sumanaweera, D., et al. (2022).

Usage

```
data("MMLSUM")
```

Format

The format is: num [1:21, 1:21, 1:9] 10.31 -10.46 -11.95 -13.59 -8.33 ... - attr(*, "dimnames")=List of 3 ..$: chr [1:21] "A" "R" "N" "D" ... ..$: chr [1:21] "A" "R" "N" "D" ... ..$: chr [1:9] "10" "20" "30" "40" ...

Details

Substitution matrix values represent the log-odds of observing an aligned pair of amino acids versus the likelihood of finding the pair by chance. The MMLSUM substitution matrices are stored as an array named by each sub-matrix's similarity threshold. (See examples section below.) In all cases values are in units of third-bits ($log(odds\ ratio) * 3/log(2)$).

Source

Sumanaweera, D., et al. (2022). Bridging the gaps in statistical models of protein alignment. *Bioinformatics*, **38(Supplement_1)**, i228-i237.

Examples

```
data(MMLSUM)
MMLSUM60 <- MMLSUM[,, "60"] # the MMLSUM60 matrix
MMLSUM60["A", "R"] # score for A/R pairing

data(BLOSUM)
plot(BLOSUM[1:20, 1:20, "62"], MMLSUM60[1:20, 1:20])
abline(a=0, b=1)
```

MODELS *Available Models of Sequence Evolution*

Description

Models of sequence evolution that are supported.

Usage

```
MODELS
```

Details

MODELS is a list of two elements: a character vector of (eight) nucleotide models and a character vector of (41) protein models. All MODELS are time reversible.

Nucleotide models are described in order of increasing number of parameters as follows:

JC69 (Jukes and Cantor, 1969) The simplest substitution model that assumes equal base frequencies (1/4) and equal mutation rates.

K80 (Kimura, 1980) Assumes equal base frequencies, but distinguishes between the rate of transitions and transversions.

T92 (Tamura, 1992) In addition to distinguishing between transitions and transversions, a parameter is added to represent G+C content bias.

F81 (Felsenstein, 1981) Assumes equal mutation rates, but allows all bases to have different frequencies.

HKY85 (Hasegawa, Kishino and Yano, 1985) Distinguishes transitions from transversions and allows bases to have different frequencies. Note that a similar model is known as **F84** (Felsenstein, 1984).

TN93 (Tamura and Nei, 1993) Allows for unequal base frequencies and distinguishes between transversions and the two possible types of transitions (i.e., A <-> G & C <-> T).

SYM (Zharkikh, 1994) Equal base frequencies but all substitution rates are free parameters.

GTR (Tavare, 1986) The general time reversible model allowing for unequal base frequencies and substitution rates.

Protein models are described in the following publications, which include both a rate matrix and corresponding frequencies:

AB (Mirsky, 2015) [Antibody]

BLOSUM62 (Henikoff, 1992) [General]

cpREV (Adachi, 2000) [Plastid]

cpREV64 (Zhong, 2010) [Plastid]

Dayhoff (Dayhoff, 1978) [General]

DCMut-Dayhoff (Kosiol, 2005) [General]

DCMut-JTT (Kosiol, 2005) [General]

DEN (Le, 2018) [Dengue virus]

FLAVI (Le, 2020) [Flavi virus]

FLU (Dang, 2010) [Influenza virus]

gcpREV (Cox, 2013) [Green plant chloroplast]

HIVb (Nickle, 2007) [Human immunodeficiency virus]

HIVin (Del Amparo and Arenas, 2022) [Human immunodeficiency virus integrase]

HIVpr (Del Amparo and Arenas, 2022) [Human immunodeficiency virus protease]

HIVw (Nickle, 2007) [Human immunodeficiency virus]

JTT (Jones, 1992) [General]

LG (Le, 2008) [General]

MtArt (Abascal, 2007) [Arthropoda mitochondrion]

mtDeu (Le, 2017) [Deuterostomia mitochondrion]

mtInv (Le, 2017) [Invertebrate mitochondrion]

mtMam (Yang, 1998) [Mammal mitochondrion]

mtMet (Le, 2017) [Metazoan mitochondrion]

mtOrt (Chang, 2020) [Orthoptera mitochondrion]

mtREV (Adachi, 1996) [Vertebrate mitochondrion]

mtVer (Le, 2017) [Vertebrate mitochondrion]

MtZoa (Rota-Stabelli, 2009) [Metazoa mitochondrion]

PMB (Veerassamy, 2003) [General]

Poisson [General]

Q.bird (Minh, 2021) [Birds]

Q.insect (Minh, 2021) [Insects]

Q.LG (Minh, 2021) [General]

Q.mammal (Minh, 2021) [Mammals]

Q.pfam (Minh, 2021) [General]

Q.plant (Minh, 2021) [Plants]

Q.yeast (Minh, 2021) [Yeast]

rtREV (Dimmic, 2002) [Retrovirus]

stmtREV (Liu, 2014) [Land plant mitochrondrion]

VIRin (Del Amparo and Arenas, 2022) [Virus integrase]

VIRpr (Del Amparo and Arenas, 2022) [Virus protease]

VT (Muller, 2000) [General]

WAG (Whelan, 2001) [General]

WAGstar (Whelan, 2001) [General]

+G (Yang, 1993) Specifying any model+G# adds a single parameter to any of the above models to relax the assumption of equal rates among sites in the sequence. The single parameter specifies

the shape of the Gamma distribution, which is represented with 2-10 discrete rates and their respective probabilities. For example, specifying a model+G8 would represent the continuous Gamma Distribution with eight rates and their associated probabilities.

+F Specifying any model+F uses fixed empirical frequencies rather than optimized or model-specific state frequencies. Note, the Poisson model is the amino acid analog of the JC69 model for nucleotides, and the Poisson+F model is the analog of F81+F.

+Indels Specifying any model+Indels adds an additional state for gaps ("-" and "." characters) and parameters for the frequency of gaps and the transition rate from non-gaps to gaps.

References

Abascal, F., Posada, D., and Zardoya, R. (2007) Molecular Biology and Evolution, **24**, 1-5.

Adachi, J. and Hasegawa, M. (1996) Journal of Molecular Evolution, **42**, 459-468.

Adachi, J., Waddell, P., Martin, W., and Hasegawa, M. (2000) Journal of Molecular Evolution, **50**, 348-358.

Chang, H., Nie, Y., Zhang, N., Zhang, X., Sun, H., Mao, Y., Qiu, Z., and Huang, Y. (2020) BMC Ecology and Evolution, **20**, 57.

Cox, C. and Foster, P. (2013) Molecular Phylogenetics and Evolution, **68**, 218-220.

Dang, C., Le, S., Gascuel, O., and Le, V. (2010) BMC Evolutionary Biology, **10**, 99.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978) Atlas of Protein Sequence and Structure, National Biomedical Research Foundation, Washington DC, **5**, 345-352.

Del Amparo, R. and Arenas, M. (2022) HIV Protease and Integrase Empirical Substitution Models of Evolution: Protein-Specific Models Outperform Generalist Models. Genes, **13(1):61**.

Dimmic, M., Rest, J., Mindell, D., and Goldstein, R. (2002) Journal of Molecular Evolution, **55**, 65-73.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution, **17(6)**, 368-376.

Felsenstein, J. (1984) A Hidden Markov Model approach to variation among sites in rate of evolution. Molecular Biology and Evolution, **13(1)**, 93-104.

Felsenstein, J. (2001) Taking Variation of Evolutionary Rates Between Sites into Account in Inferring Phylogenies. Journal of molecular evolution, **53(4-5)**, 447-455.

Hasegawa, M., Kishino H., Yano T. (1985) Dating of human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution, **22(2)**, 160-174.

Henikoff, S. and Henikoff, J. (1992) Proceedings of the National Academy of Sciences of the USA, **89**, 10915-10919.

Jones, D., Taylor, W., and Thornton, J. (1992) Computer Applications in the Biosciences, **8**, 275-282.

Jukes, T. and Cantor C. (1969) Evolution of Protein Molecules. New York: Academic Press. pp. 21-132.

Kimura, M. (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. Journal of Molecular Evolution, **16(2)**, 111-120.

Kosiol, C. and Goldman, N. (2005) Molecular Biology and Evolution, **22**, 193-199.

Le, S. and Gascuel, O. (2008) Molecular Biology and Evolution, **25**, 1307-1320.

Le, T., Dang, C., and Le, S. (2018) Proceedings of 10th International Conference on Knowledge and Systems Engineering (KSE 2018), Ho Chi Minh City, Vietnam, 242-246.

Le, T., and Vinh, L. (2020) Journal of Molecular Evolution, **88**, 445-452.

Le, V., Dang, C., and Le, S. (2017) BMC Evolutionary Biology, **17**, 136.

Liu, Y., Cox, C., Wang, W., and Goffinet, B. (2014) Systematic Biology, **63**, 862-878.

Minh, B., Dang, C., Le, S., and Lanfear, R. (2021) Systematic Biology, syab010.

Mirsky, A., Kazandjian, L., and Anisimova, M. (2015) Molecular Biology and Evolution, **32**, 806-819.

Muller, T. and Vingron, M. (2000) Journal of Computational Biology, **7**, 761-776.

Nickle, D., Heath, L., Jensen, M., Gilbert P., and Mullins, J., Kosakovsky Pond SL (2007) PLoS ONE, **2**, e503.

Rota-Stabelli, O., Yang, Z., and Telford, M. (2009) Molecular Phylogenetics and Evolution, **52**, 268-272.

Tamura, K. (1992) Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+C content biases. Molecular Biology and Evolution, **9(4)**, 678-687.

Tamura, K. and Nei M. (1993) Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. Molecular Biology and Evolution, **10(3)**, 512-526.

Tavare, S. (1986) "Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences." Lectures on Mathematics in the Life Sciences, **17**: 57-86.

Veerassamy, S., Smith, A., and Tillier, E. (2003) Journal of Computational Biology, **10**, 997-1010.

Whelan, S. and Goldman, N. (2001) Molecular Biology and Evolution, **18**, 691-699.

Yang, Z., Nielsen, R., and Hasegawa, M. (1998) Molecular Biology and Evolution, **15**, 1600-1611.

Yang, Z. (1993) Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. Molecular Biology and Evolution, **10(6)**, 1396-1401.

Zharkikh, A. (1994) Estimation of evolutionary distances between nucleotide sequences. Journal of Molecular Evolution, **39**, 315-329.

Zhong, B., Yonezawa, T., Zhong, Y., and Hasegawa, M. (2010) Molecular Biology and Evolution, **27**, 2855-2863.

See Also

`DistanceMatrix`, `Treeline`

Examples

```
str(MODELS)
```

NNLS — *Sequential Coordinate-wise Algorithm for the Non-negative Least Squares Problem*

Description

Consider the linear system $\boldsymbol{A}x = b$ where $\boldsymbol{A} \in R^{\text{m x n}}$, $x \in R^{\text{n}}$, and $b \in R^{\text{m}}$. The technique of least squares proposes to compute x so that the sum of squared residuals is minimized. NNLS solves the least squares problem $\min ||\boldsymbol{A}x = b||^2$ subject to the constraint $x \geq 0$. This implementation of the Sequential Coordinate-wise Algorithm uses a sparse input matrix $\boldsymbol{A}$, which makes it efficient for large sparse problems.

Usage

```
NNLS(A,
     b,
     precision = sqrt(.Machine$double.eps),
     processors = 1,
     verbose = TRUE)
```

Arguments

A	List representing the sparse matrix with integer components i and j, numeric component x. The fourth component, dimnames, is a list of two components that contains the names for every row (component 1) and column (component 2).
b	Numeric matrix for the set of observed values. (See details section below.)
precision	The desired accuracy.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

The input b can be either a matrix or a vector of numerics. If it is a matrix then it is assumed that each column contains a set of observations, and the output x will have the same number of columns. This allows multiple NNLS problems using the same $\boldsymbol{A}$ matrix to be solved simultaneously, and greatly accelerates computation relative to solving each sequentially.

Value

A list of two components:

x	The matrix of non-negative values that best explains the observed values given by b.
res	A matrix of residuals given by $\boldsymbol{A}x - b$.

References

Franc, V., et al. (2005). Sequential coordinate-wise algorithm for the non-negative least squares problem. Computer Analysis of Images and Patterns, 407-414.

See Also

Array2Matrix, DesignArray

Run `vignette("DesignMicroarray", package = "DECIPHER")` to see a related vignette.

Examples

```
# unconstrained least squares:
A <- matrix(c(1, -3, 2, -3, 10, -5, 2, -5, 6), ncol=3)
b <- matrix(c(27, -78, 64), ncol=1)
x <- solve(crossprod(A), crossprod(A, b))

# Non-negative least squares:
w <- which(A > 0, arr.ind=TRUE)
A <- list(i=w[,"row"], j=w[,"col"], x=A[w],
          dimnames=list(1:dim(A)[1], 1:dim(A)[2]))
x_nonneg <- NNLS(A, b)

# compare the unconstrained and constrained solutions:
cbind(x, x_nonneg$x)

# the input value "b" can also be a matrix:
b2 <- matrix(b, nrow=length(b), ncol=2) # repeat b in two columns
x_nonneg <- NNLS(A, b2) # solution is repeated in two output columns
```

NonCoding *NonCoding Objects and Methods*

Description

Non-coding RNAs can be represented by their conserved sequence motifs, secondary structure, and k-mer frequencies. Class `NonCoding` provides objects and functions for representing non-coding RNAs.

Usage

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

Arguments

`x`	An object of class `NonCoding`.
`...`	Other optional parameters.

Details

Objects of class NonCoding are stored as lists containing a compact representation of a family of non-coding RNAs. The first list component is a matrix of sequence motifs that identify the non-coding RNAs, the second is a matrix of hairpin loops that are conserved across the family, the third is a list of k-mer frequencies derived from representative sequences, and the fourth is a vector of log-odds scores for sequence lengths. An optional fifth list component denotes the log-odds scores for dependencies among patterns. Patterns are defined by their distance to either end of the non-coding RNA, which helps to identify the boundaries of the non-coding RNA in a genome.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. S. (2021). FindNonCoding: rapid and simple detection of non-coding RNAs in genomes. Bioinformatics. https://doi.org/10.1093/bioinformatics/btab708

See Also

LearnNonCoding, FindNonCoding

Examples

```
data(NonCodingRNA_Bacteria)
x <- NonCodingRNA_Bacteria
print(x)
class(x)
attributes(x[[1]])
x[[1]] # the first non-coding RNA
x[[1]][["motifs"]] # sequence motifs
x[[1]][["hairpins"]] # hairpin loops
head(x[[1]][["kmers"]]) # k-mer frequencies
```

NonCodingRNA *NonCoding Models for Common Non-Coding RNA Families*

Description

Pre-trained with NonCoding models for common RNA families found in genomes from organisms belonging to each domain of life.

Usage

```
data("NonCodingRNA_Archaea")
```

Details

A set of NonCoding models contained in a named list. Models were built from up to 1000 representative sequences per non-coding RNA family.

Source

Models were built from sequences belonging to families in tRNADB-CE (http://trna.ie.niigata-u.ac.jp/cgi-bin/trnadb/index.cgi) or Rfam (http://rfam.xfam.org).

Examples

```
data(NonCodingRNA_Archaea)
data(NonCodingRNA_Bacteria)
data(NonCodingRNA_Eukarya)
names(NonCodingRNA_Bacteria)
head(NonCodingRNA_Bacteria)
```

OrientNucleotides *Orient Nucleotide Sequences*

Description

Orients nucleotide sequences to match the directionality and complementarity of specified reference sequences.

Usage

```
OrientNucleotides(myXStringSet,
                  reference = which.max(width(myXStringSet)),
                  type = "sequences",
                  orientation = "both",
                  threshold = 0.05,
                  verbose = TRUE,
                  processors = 1)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet of unaligned sequences.
reference	The index of reference sequences with the same (desired) orientation. By default the first sequence with maximum width will be used.
type	Character string indicating the type of results desired. This should be either "sequences", "orientations", or "both".
orientation	Character string(s) indicating the allowed reorientation(s) of non-reference sequences. This should be either "all", "reverse", "complement", and/or "both" (for reverse complement).

threshold	Numeric giving the decrease in k-mer distance required to adopt the alternative orientation.
verbose	Logical indicating whether to display progress.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.

Details

Biological sequences can sometimes have inconsistent orientation that interferes with their analysis. OrientNucleotides will reorient sequences by changing their directionality and/or complementarity to match specified reference sequences in the same set, which are assume to have the same orientation. The process works by finding the k-mer distance between the reference sequence(s) and each allowed orientation of the sequences. Alternative orientations that lessen the distance by at least threshold are adopted. Note that this procedure requires a moderately similar reference sequence be available for each sequence that needs to be reoriented. Sequences for which a corresponding reference is unavailable will most likely be left alone because alternative orientations will not pass the threshold. For this reason, it is recommended to specify several diverse sequences as references.

Value

OrientNucleotides can return two types of results: the relative orientations of sequences and/or the reoriented sequences. If type is "sequences" (the default) then the reoriented sequences are returned. If type is "orientations" then a character vector is returned that specifies whether sequences were reversed ("r"), complemented ("c"), reversed complemented ("rc"), or in the same orientation ("") as the reference sequences (marked by NA). If type is "both" then the output is a list with the first component containing the "orientations" and the second component containing the "sequences".

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

CorrectFrameshifts

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna <- RemoveGaps(dna)
DNA <- dna # 175 sequences

# reorient subsamples of the first 169 sequences
s <- sample(169, 30)
DNA[s] <- reverseComplement(dna[s])
s <- sample(169, 30)
DNA[s] <- reverse(dna[s])
s <- sample(169, 30)
```

```
DNA[s] <- complement(dna[s])

DNA <- OrientNucleotides(DNA, reference=170:175)
DNA==dna # all were correctly reoriented
```

PAM *PAM Amino Acid Substitution Matrices*

Description

The PAM amino acid substitution matrices defined by Dayhoff, M., et al. (1978).

Usage

```
data("PAM")
```

Format

The format is: num [1:24, 1:24, 1:50] 10.5 -15 -10.5 -9 -15 -10.5 -7.5 -6 -16.5 -12 ... - attr(*, "dimnames")=List of 3 ..$: chr [1:24] "A" "R" "N" "D" ... ..$: chr [1:24] "A" "R" "N" "D" ... ..$: chr [1:50] "10" "20" "30" "40" ...

Details

Substitution matrix values represent the log-odds of observing an aligned pair of amino acids versus the likelihood of finding the pair by chance. The PAM substitution matrices are stored as an array named by each sub-matrix's percentage accepted mutations. (See examples section below.) In all cases values are in units of third-bits ($log(odds\ ratio) * 3/log(2)$).

Source

Dayhoff, M., et al. (1978). A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, **5**, 345-358.

Examples

```
data(PAM)
PAM250 <- PAM[,, "250"] # the PAM250 matrix
PAM250["A", "R"] # score for A/R pairing

data(BLOSUM)
plot(BLOSUM[1:20, 1:20, "62"], PAM250[1:20, 1:20])
abline(a=0, b=1)
```

PFASUM *PFASUM Amino Acid Substitution Matrices*

Description

The PFASUM amino acid substitution matrices defined by Keul, F., et al. (2017).

Usage

```
data("PFASUM")
```

Format

The format is: num [1:25, 1:25, 1:90] 0.9492 -1.7337 0.2764 1.8153 0.0364 ... - attr(*, "dimnames")=List of 3 ..$: chr [1:25] "A" "R" "N" "D" ... ..$: chr [1:25] "A" "R" "N" "D" ... ..$: chr [1:90] "11" "12" "13" "14" ...

Details

Substitution matrix values represent the log-odds of observing an aligned pair of amino acids versus the likelihood of finding the pair by chance. The PFASUM substitution matrices are stored as an array named by each sub-matrix's similarity threshold. (See examples section below.) In all cases values are in units of third-bits ($log(odds\ ratio) * 3/log(2)$).

Source

Keul, F., et al. (2017). PFASUM: a substitution matrix from Pfam structural alignments. *BMC Bioinformatics*, **18(1)**, 293.

Examples

```
data(PFASUM)
PFASUM62 <- PFASUM[,, "62"] # the PFASUM62 matrix
PFASUM62["A", "R"] # score for A/R pairing

data(BLOSUM)
plot(BLOSUM[1:20, 1:20, "62"], PFASUM62[1:20, 1:20])
abline(a=0, b=1)
```

PredictDBN *Predict RNA Secondary Structure in Dot-Bracket Notation*

Description

Predicts a consensus RNA secondary structure from a multiple sequence alignment using mutual information.

Usage

```
PredictDBN(myXStringSet,
           type = "states",
           minOccupancy = 0.4,
           impact = c(1, 1.2, 0.4, -1),
           avgProdCorr = 1,
           slope = 2,
           shift = 1.3,
           threshold = 0.4,
           pseudoknots = 1,
           weight = NA,
           useFreeEnergy = TRUE,
           deltaGrules = NULL,
           processors = 1,
           verbose = TRUE)
```

Arguments

myXStringSet	A DNAStringSet or RNAStringSet object containing aligned sequences.
type	Character string indicating the type of results desired. This should be (an unambiguous abbreviation of) one of "states", "pairs", "evidence", "scores", "structures", or "search". (See value section below.)
minOccupancy	Numeric specifying the minimum occupancy (1 - fraction of gaps) required to include a column of the alignment in the prediction.
impact	A vector with four elements giving the weights of A/U, G/C, G/U, and other pairings, respectively. The last element of impact is the penalty for pairings that are inconsistent with two positions being paired (e.g., A/- or A/C).
avgProdCorr	Numeric specifying the weight of the average product correction (APC) term, as described in Buslje et al. (2009).
slope	Numeric giving the slope of the sigmoid used to convert mutual information values to scores ranging from zero to one.
shift	Numeric giving the relative shift of the sigmoid used to convert mutual information values to scores ranging from zero to one.
threshold	Numeric specifying the score threshold at which to consider positions for pairing. Only applicable if type is "states" or "pairs".

pseudoknots	Integer indicating the maximum order of pseudoknots that are acceptable. A value of 0 will prevent pseudoknots in the structure, whereas 1 (the default) will attempt to find first-order psuedoknots. Only used if type is "states" or "pairs".
weight	Either a numeric vector of weights for each sequence, a single number implying equal weights, or NA (the default) to automatically calculate sequence weights based on myXStringSet.
useFreeEnergy	Logical determining whether RNA free energy predictions should be incorporated along with mutual information into the secondary structure prediction.
deltaGrules	Free energy rules for all possible base pairings in quadruplets. If NULL, defaults to pseudoenergies (deltaGrulesRNA). Only applicable if useFreeEnergies is TRUE.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.

Details

PredictDBN employs an extension of the method described by Freyhult et al. (2005) for determining a consensus RNA secondary structure. It uses the mutual information (H) measure to find covarying positions in a multiple sequence alignment. The original method is modified by the addition of different weights for each type of base pairing and each input sequence. The formula for mutual information between positions i and j then becomes:

$$H(i,j) = \sum_{XY \in bp} \left(impact(XY) \cdot f_{i,j}(XY) \cdot \log_2 \left(\frac{f_{i,j}(XY)}{f_i(X) \cdot f_j(Y)} \right) \right)$$

where, bp denotes the base pairings A/U, C/G, and G/U; impact is their weight; f is the frequency of single bases or pairs weighted by the corresponding weight of each sequence.

A penalty is then added for bases that are inconsistent with pairing:

$$H_{mod}(i,j) = H(i,j) + \sum_{XY \notin bp} \Big(impact(XY) \cdot f_{i,j}(XY) \Big)$$

Next an average product correction (Buslje et al., 2009) is applied to the matrix H:

$$H_{APC}(i,j) = H_{mod}(i,j) - avgProdCorr \cdot \frac{\overline{H_{mod}(i,.)} \cdot \overline{H_{mod}(.,j)}}{\overline{H_{mod}(.,.)}}$$

The mutual information values are then rescaled between 0 and 1 by applying a sigmoidal transformation, which is controlled by shift and slope:

$$H_{final}(i,j) = \left(1 + \exp \left(slope \cdot log_e \left(\frac{H_{APC}(i,j)}{shift \cdot H_{APC}[n]} \right) \right) \right)^{-1}$$

where, n is the number of positions having `minOccupancy` divided by two (i.e., the maximum possible number of paired positions) and $H_{APC}[n]$ denotes the n^{th} highest value in the matrix H_{APC}.

If `useFreeEnergies` is `TRUE`, mutual information is supplemented with a probabalistic model of folding based on `deltaGrules`. That is, palindromes in each sequence are ranked by their free energy, and converted to probabilities of base pairing by assuming an exponential distribution of free energies. This tends to improve predictive accuracy when the aligned sequences are insufficiently diverse for considerable evidence of compensatory mutations.

If `type` is `"states"` or `"pairs"`, the secondary structure is determined using a variant of the Nussinov algorithm similar to that described by Venkatachalam et al. (2014). Pairings with a score below `threshold` are not considered during the traceback. If `psuedoknots` is greater than `0`, paired positions are removed from consideration and the method is applied again to find pseudoknots.

In practice the secondary structure prediction is most accurate when the input alignment is of high quality, contains a wide diversity of sequences, the number of sequences is large, no regions are completely conserved across all sequences, and most of the sequences span the entire alignment (i.e., there are few partial/incomplete sequences).

Value

If `type` is `"states"` (the default), then the output is a character vector with the predicted secondary structure assignment for each position in `myXStringSet`. Standard dot-bracket notation (DBN) is used, where "." signifies an unpaired position, "(" and ")" a paired position, and successive "[]", "{}", and "<>" indicate increasing order pseudoknots. Columns below `minOccupancy` are denoted by the "-" character to indicate that they contained too many gaps to be included in the consensus structure.

If `type` is `"pairs"`, then a matrix is returned with one row for each base pairing and three columns giving the positions of the paired bases and their `pseudoknot` order.

If `type` is `"evidence"`, then a matrix is returned with one row for each base pairing and three columns giving the positions of the paired bases and their respective scores (greater than or equal to `threshold`). This differs from `type` `"pairs"` in that `"evidence"` does not perform a traceback. Therefore, it is possible to have conflicting evidence where a single base has evidence for pairing with multiple other bases.

If `type` is `"scores"`, then a matrix of three rows is returned, where the values in a column represent the maximum score for a state in each position. Columns sum to `1` if the position was above `minOccupancy` and `0` otherwise.

If `type` is `"structures"`, then the output is a list with one element for each sequence in `myXStringSet`. Each list element contains a matrix of dimension 3 (each state) by the number of nucleotides in the sequence. Columns of the matrix sum to zero where the nucleotide was located in a position that was below `minOccupancy`. Otherwise, positions are considered paired if they are consistent with pairing (i.e., A/U, C/G, or G/U) in the consensus secondary structure.

When `type` is `"search"` the results are similar to `"structures"`, but an attempt is made to find additional secondary structure beyond positions exhibiting covariation. First, anchors are identified as pairs of covarying positions with their score above `threshold`. Next, the regions between anchors are searched for previously unidentified stem loops. Finally, any helices are assigned a score according to their length, i.e. one minus the probability of finding that many consecutive pairs within

the anchor boundaries by chance. Hence, output `type "search"` will find secondary structure outside of the consensus structure shared by most sequences, and can identify secondary structure in conserved alignment regions.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Buslje, C., *et al.* (2009). Correction for phylogeny, small number of observations and data redundancy improves the identification of coevolving amino acid pairs using mutual information. Bioinformatics, **25(9)**, 1125-1131.

Freyhult, E., *et al.* (2005). Predicting RNA Structure Using Mutual Information. Applied Bioinformatics, **4(1)**, 53-59.

Venkatachalam, B., *et al.* (2014). Faster algorithms for RNA-folding using the Four-Russians method. Algorithms for Molecular Biology : AMB, **9(1)**, 1-12.

Wright, E. S. (2020). RNAconTest: comparing tools for noncoding RNA multiple sequence alignment based on structural consistency. RNA 2020, 26, 531-540.

See Also

PredictHEC

Run `vignette("FindingNonCodingRNAs", package = "DECIPHER")` to see a related vignette.

Examples

```
# load the example non-coding RNA sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
rna <- RNAStringSet(dna)

## Not run:
# predict the secondary structure in dot-bracket notation (dbn)
p <- PredictDBN(rna, "states") # predict the secondary structure in dbn
p # pairs are denoted by (), and (optionally) pseudoknots by [], {}, and <>

# convert the dot-bracket notation into pairs of positions within the alignment
p <- PredictDBN(rna, "pairs") # paired positions in the alignment
head(p) # matrix giving the pairs and their pseudoknot order (when > 0)

# plot an arc diagram with the base pairings
plot(NA, xlim=c(0, 1), ylim=c(0, 1),
 xaxs="i", yaxs="i",
 xlab="Alignment position", ylab="",
 bty="n", xaxt="n", yaxt="n")
ticks <- pretty(seq_len(width(rna)[1]))
axis(1, ticks/width(rna)[1], ticks)
rs <- c(seq(0, pi, len=100), NA)
r <- (p[, 2] - p[, 1] + 1)/width(rna)[1]/2
```

```
r <- rep(r, each=101)
x <- (p[, 1] + p[, 2])/2/width(rna)[1]
x <- rep(x, each=101) + r*cos(rs)
y <- r*sin(rs)/max(r, na.rm=TRUE)
lines(x, y, xpd=TRUE)

# show all available evidence of base pairing
p <- PredictDBN(rna, "evidence") # all pairs with scores >= threshold
head(p) # matrix giving the pairs and their scores

# determine the score at every alignment position
p <- PredictDBN(rna, "scores") # score in the alignment
p["(", 122] # score for left-pairing at alignment position 122
p[")", 260] # score for right-pairing at alignment position 260

# find the scores individually for every sequence in the alignment
p <- PredictDBN(rna, "structures") # scores per sequence
p[[1]][, 1] # the scores for the first position in the first sequence
p[[2]][, 10] # the scores for the tenth position in the second sequence
# these positional scores can be used as shades of red, green, and blue:
BrowseSeqs(rna, patterns=p) # red = unpaired, green = left-pairing, blue = right
# positions in black are not part of the consensus secondary structure

# search for additional secondary structure between the consensus pairs
p <- PredictDBN(rna, "search") # scores per sequence after searching
BrowseSeqs(rna, patterns=p) # red = unpaired, green = left-pairing, blue = right
# note that "search" identified many more pairings than "structures"

## End(Not run)
```

PredictHEC *Predict Protein Secondary Structure*

Description

Predicts protein secondary structure based on the primary (amino acid) sequence using the GOR IV method (Garnier et al., 1996).

Usage

```
PredictHEC(myAAStringSet,
           type = "states",
           windowSize = 8,
           background = c(H = -0.2, E = -0.6, C = 0),
           HEC_MI1 = NULL,
           HEC_MI2 = NULL)
```

Arguments

myAAStringSet	An AAStringSet object of sequences.
type	Character string indicating the type of results desired. This should be (an unambiguous abbreviation of) one of "states", "scores", or "probabilities".
windowSize	Either a single number specifying the number of residues to the left or right of the center position to use in the prediction or two numbers to use with HEC_MI1 and HEC_MI2, respectively.
background	A named numeric vector with the background "scores" for each of the states, or a matrix with states as rows and columns giving the background for each sequence in myXStringSet. The background represents a calibrated prior on the relative propensity for each state, typically the log of the (normalized) state frequencies and possibly standardized to a minimum or maximum of zero.
HEC_MI1	An array of dimensions 20 x W x N giving the mutual information for single residues, where W is at least 2*windowSize + 1 and N is the number of states in background.
HEC_MI2	An array of dimensions 20 x 20 x W x W x N giving the mutual information for pairs of residues, where W is at least 2*windowSize + 1 and N is the number of states in background.

Details

The GOR (Garnier-Osguthorpe-Robson) method is an information-theory method for prediction of secondary structure based on the primary sequence of a protein. Version IV of the method makes state predictions based on the mutual information contained in single residues and pairs of residues within windowSize residues of the position being assigned. This approach is about 65% accurate for the traditional three states (i.e., H, E, and C), and is one of the more accurate methods for assigning secondary structure based on single sequences. This implementation of GOR IV does not use decision constants or the number of contiguous states when assigning the final state. Note that characters other than the standard 20 amino acids are not assigned a state.

Value

If type is "states" (the default), then the output is a character vector with the secondary structure assignment for each residue in myAAStringSet.

Otherwise, the output is a list with one element for each sequence in myAAStringSet. Each list element contains a matrix of dimension N by the number of residues in the sequence, where N is the number of states in background. If type is "scores", then values in the matrix represent log-odds "scores". If type is "probabilities" then the values represent the normalized probabilities of the states at a position.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Garnier, J., Gibrat, J. F., & Robson, B. (1996). GOR method for predicting protein secondary structure from amino acid sequence. *Methods in Enzymology*, **266**, 540-553.

See Also

HEC_MI1, HEC_MI2, PredictDBN

Examples

```
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
aa <- translate(dna)
hec <- PredictHEC(aa)
head(hec)
```

ReadDendrogram *Read a Dendrogram from a Newick Formatted File*

Description

Reads a dendrogram object from a file in Newick (also known as New Hampshire) parenthetic format.

Usage

```
ReadDendrogram(file,
               convertBlanks = TRUE,
               internalLabels = TRUE,
               keepRoot = TRUE,
               quote = "'")
```

Arguments

file	A connection object, a file path, or a character string providing the contents of a Newick formatted file.
convertBlanks	Logical specifying whether to convert underscores in unquoted leaf labels to spaces.
internalLabels	Logical indicating whether to keep internal node labels as "edgetext" preceding the node in the dendrogram.
keepRoot	Logical specifying whether to keep the root node (if one is present) as a dendrogram leaf.
quote	Either a single or double quotation mark determining the character used quote labels.

Details

ReadDendrogram will create a dendrogram object from a Newick formatted tree. Note that all edge lengths must be specified, but labels are optional. Leaves will be numbered by their labels in alphabetical order. Multiple dendrograms can be read if separated by semicolons (i.e., ";"), in which case the output is a list containing dendrograms.

Value

An object of class dendrogram or a list of dendrograms.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. (2026) Treeline Provides a Unified Strategy for Optimising Phylogenetic Trees Under Alternative Criteria. *Molecular Ecology Resources*, **26(5)**, e70163.

See Also

Treeline, WriteDendrogram

Run vignette("GrowingTrees", package = "DECIPHER") to see a related vignette.

Examples

```
tf <- tempfile()
dists <- matrix(c(0, 10, 20, 10, 0, 5, 20, 5, 0),
    nrow=3,
    dimnames=list(c("dog", "elephant", "horse")))
dend1 <- Treeline(myDistMatrix=dists, method="NJ", type="dendrogram")
WriteDendrogram(dend1, file=tf)

dend2 <- ReadDendrogram(tf) # returns a dendrogram
layout(matrix(1:2))
plot(dend1, main="Dendrogram Written")
plot(dend2, main="Dendrogram Read")

# read a multi-dendrogram Newick file
WriteDendrogram(dend1, file=tf, append=TRUE) # add another dendrogram
ReadDendrogram(tf) # now a list of dendrograms

unlink(tf)
```

RemoveGaps *Remove Gap Characters in Sequences*

Description

Removes gaps ("-" or "." characters) in a set of sequences, either deleting all gaps or only those shared by all sequences in the set.

Usage

```
RemoveGaps(myXStringSet,
           removeGaps = "all",
           includeMask = FALSE,
           processors = 1)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object containing sequences.
removeGaps	Determines how gaps ("-" or "." characters) are removed in the sequences. This should be (an unambiguous abbreviation of) one of "none", "all" or "common".
includeMask	Logical specifying whether to consider the mask character ("+") as a gap.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.

Details

The removeGaps argument controls which gaps are removed in myXStringSet. Setting removeGaps to "all" will remove all gaps in the input sequences, whereas setting removeGaps to "common" will remove only gaps that exist in the same position in every sequence. Therefore, the latter method will leave gaps in place that are not shared by every sequence, requiring that the sequences in myXStringSet all have the same width (i.e., be aligned). Setting removeGaps to "none" will simply return myXStringSet unaltered.

Value

An XStringSet of the same type as myXStringSet.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

AlignSeqs

Examples

```
dna <- DNAStringSet(c("ACT-G-", "AC--G-"))
dna
RemoveGaps(dna, "all")
RemoveGaps(dna, "common")
```

RESTRICTION_ENZYMES *Common Restriction Enzyme's Cut Sites*

Description

A character vector of common restriction sites named by the restriction enzyme that cuts at each site. Sequence specificity is listed in 5' to 3' orientation based on the IUPAC_CODE_MAP. The cut site is either signified by a "/" for palindromic sites, or two numbers giving the position of the top and bottom cut positions relative to the site's 3'-end.

Usage

```
data(RESTRICTION_ENZYMES)
```

Format

The format is: Named chr [1:224] "GACGT/C" "G/GTACC" "GT/MKAC" ... - attr(*, "names")= chr [1:224] "AatII" "Acc65I" "AccI" "AciI" ...

Source

Restriction enzymes sold by New England BioLabs.

Examples

```
data(RESTRICTION_ENZYMES)
RESTRICTION_ENZYMES
```

ScoreAlignment *Score a Multiple Sequence Alignment*

Description

Calculates a score for a multiple sequence alignment based on either sum-of-pairs or sum-of-adjacent-pairs scoring.

Usage

```
ScoreAlignment(myXStringSet,
               method = "pairs",
               type = "sum",
               perfectMatch = 1,
               misMatch = 0,
               gapOpening = -7.5,
               gapExtension = -0.6,
               substitutionMatrix = NULL,
```

```
structures = NULL,
structureMatrix = NULL,
includeTerminalGaps = FALSE,
weight = 1)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of aligned sequences.
method	Character string indicating the method of scoring. This should be one of "pairs" for sum-of-pairs or "adjacent" for sum-of-adjacent-pairs. (See details section below.)
type	Character string giving the type of result. This should be one of "sum" for the total score or "scores" for a vector with one score per site (column) in myXStringSet.
perfectMatch	Numeric giving the reward for aligning two matching nucleotides in the alignment. Only used for DNAStringSet or RNAStringSet inputs.
misMatch	Numeric giving the cost for aligning two mismatched nucleotides in the alignment. Only used for DNAStringSet or RNAStringSet inputs.
gapOpening	Numeric giving the cost for opening and closing a gap in the alignment.
gapExtension	Numeric giving the cost for extending an open gap in the alignment.
substitutionMatrix	Either a substitution matrix representing the substitution scores for an alignment (in third-bits) or the name of the amino acid substitution matrix to use in alignment. The default (NULL) will use the perfectMatch and misMatch penalties for DNA/RNA or PFASUM50 for AA.
structures	Either NULL (the default) or a list of matrices with one list element per sequence in myXStringSet.
structureMatrix	A structure matrix if structures are supplied, or NULL otherwise.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" or "." characters on each end of the sequence) into the calculation of score.
weight	A numeric vector of weights for each sequence, or a single number implying equal weights.

Details

Sum-of-pairs scoring is the standard way to judge whether a set of sequences are homologous. ScoreAlignment calculates the sum-of-pairs score for myXStringSet when method is "pairs". This score can also be used to compare among different alignments of the same sequences. If method is "adjacent" then the sum-of-adjacent-pairs scores is calculated, where each sequence is compared to the next sequence. Hence, the input order of sequences in myXStringSet matters when method is "adjacent".

Both scores are linearly related to the number of sequences in the alignment and the number of sites in the alignment. Therefore, it is possible to normalize the score by dividing by the width and length (minus 1) of the myXStringSet.

Value

A single numeric score.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

AlignSeqs, PFASUM

Examples

```
# small example
x <- DNAStringSet(c("C-G", "CTG", "C-G", "CTG"))
ScoreAlignment(x, method="pairs", gapOpening=-1) # +3 -1 +3 = 5
ScoreAlignment(x, method="adjacent", gapOpening=-1) # +3 -3 +3 = 3

# DNA alignment with the defaults
fas <- system.file("extdata", "Streptomyces_ITS_aligned.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna # input alignment
ScoreAlignment(dna, method="pairs")
ScoreAlignment(dna, method="adjacent")

# provide a DNA substitution matrix for greater discerning power
sub <- matrix(c(1.5, -2.134, -0.739, -1.298,
  -2.134, 1.832, -2.462, 0.2,
  -0.739, -2.462, 1.522, -2.062,
  -1.298, 0.2, -2.062, 1.275),
 nrow=4,
 dimnames=list(DNA_BASES, DNA_BASES))
ScoreAlignment(dna, substitutionMatrix=sub)

# use structures with an amino acid alignment
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
aa <- AlignTranslation(dna, type="AAStringSet")
structureMatrix <- matrix(c(0.187, -0.8, -0.873,
  -0.8, 0.561, -0.979,
  -0.873, -0.979, 0.221),
 nrow=3,
 dimnames=list(c("H", "E", "C"), c("H", "E", "C")))
ScoreAlignment(aa,
 structures=PredictHEC(aa, type="probabilities"),
 structureMatrix=structureMatrix)
```

SearchDB *Obtain Specific Sequences from a Database*

Description

Returns the set of sequences meeting the search criteria.

Usage

```
SearchDB(dbFile,
         tblName = "Seqs",
         identifier = "",
         type = "XStringSet",
         limit = -1,
         replaceChar = NA,
         nameBy = "row_names",
         orderBy = "row_names",
         countOnly = FALSE,
         removeGaps = "none",
         quality = "Phred",
         clause = "",
         processors = 1,
         verbose = TRUE)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table where the sequences are located.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" (the default) then all identifiers are selected.
type	The type of XStringSet (sequences) to return. This should be (an unambiguous abbreviation of) one of "XStringSet", "DNAStringSet", "RNAStringSet", "AAStringSet", "BStringSet", "QualityScaledXStringSet", "QualityScaledDNAStringSet", "QualityScaledRNAStringSet", "QualityScaledAAStringSet", or "QualityScaledBStringSet". If type is "XStringSet" or "QualityScaledXStringSet" then an attempt is made to guess the type of sequences based on their composition.
limit	Number of results to display. The default (-1) does not limit the number of results.
replaceChar	Optional character used to replace any characters of the sequence that are not present in the XStringSet's alphabet. Not applicable if type=="BStringSet". The default (NA) results in an error if an incompatible character exist. (See details section below.)
nameBy	Character string giving the column name for naming the XStringSet.

orderBy	Character string giving the column name for sorting the results. Defaults to the order of entries in the database. Optionally can be followed by " ASC" or " DESC" to specify ascending (the default) or descending order.
countOnly	Logical specifying whether to return only the number of sequences.
removeGaps	Determines how gaps ("-" or "." characters) are removed in the sequences. This should be (an unambiguous abbreviation of) one of "none", "all" or "common".
clause	An optional character string to append to the query as part of a "where clause".
quality	The type of quality object to return if type is a QualityScaledXStringSet. This should be (an unambiguous abbreviation of) one of "Phred", "Solexa", or "Illumina". Note that recent versions of Illumina software provide "Phred" formatted quality scores.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display queries as they are sent to the database.

Details

If type is "DNAStringSet" then all U's are converted to T's before creating the DNAStringSet, and vice versa if type is "RNAStringSet". All remaining characters not in the XStringSet's alphabet are converted to replaceChar or removed if replaceChar is "". Note that if replaceChar is NA (the default), it will result in an error when an unexpected character is found. Quality information is interpreted as PredQuality scores.

Value

An XStringSet or QualityScaledXStringSet with the sequences that meet the specified criteria. The names of the object correspond to the value in the nameBy column of the database.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

See Also

Seqs2DB, DB2Seqs

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 # get all sequences in the default table:
 dna <- SearchDB(db)
 # remove gaps from "Sphingomonadales" sequences:
```

```
 dna <- SearchDB(db, identifier="Sphingomonadales", removeGaps="all")
 # provide a more complex query:
 dna <- SearchDB(db, nameBy="description", orderBy="standard", removeGaps="common",
                 clause="nonstandard is 0")
}
```

SearchIndex *Search an inverted index*

Description

Searches an inverted index for homologous sequences.

Usage

```
SearchIndex(pattern,
            invertedIndex,
            subject=NULL,
            minScore = NA,
            perPatternLimit=0,
            perSubjectLimit=1,
            scoreOnly = FALSE,
            sepCost = -0.9,
            gapCost = -3,
            maskRepeats = TRUE,
            maskLCRs = TRUE,
            dropScore = -10,
            correctBackground = TRUE,
            iterations = 0,
            threshold = 1e-6,
            processors = 1,
            verbose = TRUE)
```

Arguments

pattern	An AAStringSet, DNAStringSet, or RNAStringSet object of query (unaligned) sequences to use as the pattern that will be queried.
invertedIndex	An object of class InvertedIndex compatible with the class of pattern.
subject	The XStringSet object used to build the invertedIndex or NULL (the default) to skip the extension of k-mer matches. (See details section below.)
minScore	Numeric specifying the minimum score of hits to return. The default (NA) will automatically determine minScore from the size of the invertedIndex to correct for multiple testing (i.e., searching more than one subject sequence).
perPatternLimit	Numeric giving the maximum number of hits to return per pattern query. The default perPatternLimit (0) will cause the number of hits per pattern to be unlimited. Note, perPatternLimit is enforced after applying the perSubjectLimit.

perSubjectLimit
: Numeric determining the maximum number of hits per subject to return for each pattern query. The default (1) is to return the top scoring hit per subject (target) sequence. Setting perSubjectLimit to 0 will cause the number of hits per subject to be unlimited. Note, perSubjectLimit is enforced before applying the perPatternLimit.

scoreOnly
: Logical determining whether to return only the hits and their scores or also the Position of k-mer hits.

sepCost
: Numeric giving the penalty applied to sequence positions separating neighboring k-mer hits.

gapCost
: Numeric providing the penalty applied to the minimum number of implied inserted or deleted positions (i.e., gaps) separating neighboring k-mer hits.

maskRepeats
: Logical specifying whether to mask repeats when searching for hits.

maskLCRs
: Logical indicating whether to mask low complexity regions when searching for hits.

dropScore
: Numeric giving the decrease from maximum score required to stop extending k-mer matches. Only applicable when subject is not NULL. Values closer to zero will increase speed, potentially at the expense of sensitivity.

correctBackground
: Logical determining whether to correct the substitution matrix for the background distribution of letter frequencies on a per pattern basis. Alternatively, a positive numeric dictating the degree of compositional adjustment, with larger values resulting in less adjustment. The default (TRUE) uses an empirically determined value of 1e4.

iterations
: The number of profile-based iterations to perform, or 0 (the default) for none. Additional iterations make use of previous homologous matches to detect more remote homologs, but each iteration increases the search time.

threshold
: Numeric controlling profile inclusion when iterations is non-zero. Should be a positive number representing the score probability required for matches to be included in the profile. Values closer to zero will generate less diverse profiles when iterating.

processors
: The number of processors to use, or NULL to automatically detect and use all available processors.

verbose
: Logical indicating whether to display progress.

Details

The invertedIndex is searched for all umasked k-mers shared with pattern, and the set of matches meeting the minScore is returned. By default, SearchIndex returns the top match per subject (target) sequence (above minScore), but it is also possible to set (or remove) a limit on the number of pattern hits (perPatternLimit) or subject hits (perSubjectLimit) output for each pattern (query) sequence. A data.frame is returned with (by default) or without the Position(s) of matches, depending on the value of scoreOnly.

If the set of subject sequences is provided (i.e., not NULL), then k-mer matches are extended to increase search sensitivity. Extension proceeds to the left and right of each k-mer match until another

match is encountered or the score falls below dropScore. This can decrease search speed, depending on dropScore, but may help to find more distant matches. Similarly, increasing the number of iterations can improve homolog detection at the expense of search speed. Each additional iteration will build a position-specific profile from significant matches and use this profile to find new hits. The Score of any hits is defined by their log-odds with respect to the pattern regardless of whether subject is provided or iterations is above zero.

Value

A data.frame is returned with dimensions with columns Pattern, Subject, Score, and (optionally) Position. The Pattern is the index of the sequence in pattern and the Subject is the index of the sequence in the set used to build the invertedIndex. Each row contains a hit with Score meeting the minScore. If scoreOnly is FALSE (the default), the Position column contains a list of matrices with four rows: start/end positions of k-mer hits in the Pattern and start/end positions of k-mer hits in the Subject. The data.frame will always be order by ascending Pattern index.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2024) "Fast and Flexible Search for Homologous Biological Sequences with DECIPHER v3". The R Journal, **16(2)**, 191-200.

See Also

IndexSeqs, AlignPairs

Run vignette("SearchForResearch", package = "DECIPHER") to see a related vignette.

Examples

```
# import target sequences
fas <- system.file("extdata", "PlanctobacteriaNamedGenes.fas.gz", package="DECIPHER")
target <- readAAStringSet(fas)

# build an inverted index
index <- IndexSeqs(target, K=6)
index

# import query sequences
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
query <- translate(dna)

# search the index, using the defaults
hits <- SearchIndex(query, index)
head(hits)
dim(hits) # number of hits

# search the index, requesting only the top hits
```

```
tophits <- SearchIndex(query, index, perPatternLimit=1)
head(tophits)
dim(tophits) # number of hits
tophits$Position[[1]] # query/target k-mer positions supporting first hit

# search the index, requesting the score for all hits
allhits <- SearchIndex(query, index, perSubjectLimit=0, scoreOnly=TRUE)
head(allhits)
dim(allhits) # number of hits

# include the target sequences to improve sensitivity (but slower)
morehits <- SearchIndex(query, index, target)
head(morehits)
dim(morehits) # number of hits

# further improve sensitivity with profile-based iterations
morehits <- SearchIndex(query, index, target, iterations=1)
head(morehits)
dim(morehits) # number of hits
```

Seqs2DB *Add Sequences from Text File to Database*

Description

Adds sequences to a database.

Usage

```
Seqs2DB(seqs,
        type,
        dbFile,
        identifier,
        tblName = "Seqs",
        chunkSize = 1e7,
        replaceTbl = FALSE,
        fields = c(accession = "ACCESSION", organism = "ORGANISM"),
        processors = 1,
        verbose = TRUE,
        ...)
```

Arguments

seqs	A connection object or a character string specifying the file path to the file containing the sequences, an XStringSet object if type is XStringSet, or a QualityScaledXStringSet object if type is QualityScaledXStringSet. Files compressed with gzip, bzip2, xz, or lzma compression are automatically detected and decompressed during import. Full URL paths (e.g., "http://" or "ftp://") to uncompressed text files or gzip compressed text files can also be used.

type	The type of the sequences (seqs) being imported. This should be (an unambiguous abbreviation of) one of "FASTA", "FASTQ", "GenBank", "XStringSet", or "QualityScaledXStringSet".
dbFile	A database connection object or a character string specifying the path to a SQLite database file. If the tblName does not exist then a new table is created (and file, if needed).
identifier	Character string specifying the "id" to give the imported sequences in the database.
tblName	Character string specifying the table in which to add the sequences.
chunkSize	Number of characters to read at a time.
replaceTbl	Logical indicating whether to overwrite the entire table in the database. If FALSE (the default) then the sequences are appended to any already existing in the tblName. If TRUE the entire table is dropped, removing any existing sequences before adding any new sequences.
fields	Named character vector providing the fields to import from a "GenBank" formatted file as text columns in the database (not applicable for other "type"s). The default is to import the "ACCESSION" field as a column named "accession" and the "ORGANISM" field as a column named "organism". Other uppercase fields, such as "LOCUS" or "VERSION", can be specified in similar manner. Note that the "DEFINITION" field is automatically imported as a column named "description" in the database.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display each query as it is sent to the database.
...	Further arguments to be passed directly to Codec for compressing sequence information.

Details

Sequences are imported into the database in chunks of lines specified by chunkSize. The sequences can then be identified by searching the database for the identifier provided. Sequences are added to the database verbatim, so that no sequence information is lost when the sequences are exported from the database. The sequence (record) names are recorded into a column named "description" in the database.

Value

The total number of sequences in the database table is returned after import.

Warning

If replaceTbl is TRUE then any sequences already in the table are overwritten, which is equivalent to dropping the entire table.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

ES Wright (2016) "Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R". The R Journal, **8(1)**, 352-359.

See Also

BrowseDB, SearchDB, DB2Seqs

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
 dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
 Seqs2DB(gen, "GenBank", dbConn, "Bacteria")
 BrowseDB(dbConn)
 dna <- SearchDB(dbConn, nameBy="description")
 dbDisconnect(dbConn)
}
```

StaggerAlignment *Produce a Staggered Alignment*

Description

Staggers overlapping characters in a multiple sequence alignment that are better explained by multiple insertions than multiple deletions.

Usage

```
StaggerAlignment(myXStringSet,
                 tree = NULL,
                 threshold = 2,
                 fullLength = FALSE,
                 processors = 1,
                 verbose = TRUE,
                 ...)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet object of aligned sequences.
tree	A bifurcating dendrogram representing the evolutionary relationships between sequences. The root should be the topmost node of the tree. The default (NULL) will automatically infer a tree from myXStringSet using Treeline.
threshold	Numeric giving the ratio of insertions to deletions that must be met to stagger a region of the alignment. Specifically, the number of insertions divided by deletions must be less than threshold to stagger. Lower values are less likely to stagger, and higher values are more likely to stagger.

fullLength	Logical specifying whether the sequences are full-length (TRUE), or terminal gaps should be treated as missing data (FALSE, the default). Either a single logical, a vector with one logical per sequence, or a list with right and left components containing logicals for the right and left sides of the alignment.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Additional arguments to be passed directly to Treeline.

Details

Multiple sequence aligners typically maximize true homologies at the expense of increased false homologies. StaggerAlignment creates a "staggered alignment" which separates regions of the alignment that are likely not homologous into separate regions. This re-balances the trade-off between true positives and false positives by decreasing the number of false homologies at the loss of some true homologies. The resulting alignment is less aesthetically pleasing because it is widened by the introduction of many gaps. However, in an evolutionary sense a staggered alignment is more correct because each aligned position represents a hypothesis about evolutionary events: overlapping characters between any two sequences represent positions common to their ancestor sequence that may have evolved through substitution.

The single parameter threshold controls the degree of staggering. Its value represents the ratio of insertions to deletions that must be crossed in order to stagger a region. A threshold of 1 would mean any region that could be better explained by separate insertions than deletions should be staggered. A higher value for threshold makes it more likely to stagger, and vice versa. A very high value would conservatively stagger most regions with gaps, resulting in few false homologies but also fewer true homologies. The default value (3) is intended to remove more false homologies than it eliminates in true homologies. It may be preferable to tailor the threshold depending on the purpose of the alignment, as some downstream procedures (such as tree building) may be more or less sensitive to false homologies.

Value

An XStringSet of aligned sequences.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

AdjustAlignment, AlignSeqs, Treeline

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
dna <- RemoveGaps(dna)
alignedDNA <- AlignSeqs(dna)
staggerDNA <- StaggerAlignment(alignedDNA)
```

```
BrowseSeqs(staggerDNA, highlight=1)
```

Synteny *Synteny blocks and hits*

Description

Syntenic blocks are DNA segments composed of conserved hits occurring in the same order on two sequences. The two sequences are typically chromosomes of different species that are hypothesized to contain homologous regions. Class "Synteny" provides objects and functions for storing and viewing syntenic blocks and hits that are shared between sequences.

Usage

```
## S3 method for class 'Synteny'
pairs(x,
     bounds = TRUE,
     boxBlocks = FALSE,
     labels = abbreviate(rownames(x), 9),
     gap = 0.5,
     line.main = 3,
     cex.labels = NULL,
     font.labels = 1,
     ...)

## S3 method for class 'Synteny'
plot(x,
     colorBy = 1,
     colorRamp = colorRampPalette(c("#FCF9EE", "#FFF272",
                                    "#FFAC28", "#EC5931",
                                    "#EC354D", "#0D0887")),
     barColor = "#CCCCCC",
     barSides = ifelse(nrow(x) < 100, TRUE, FALSE),
     horizontal = TRUE,
     labels = abbreviate(rownames(x), 9),
     cex.labels = NULL,
     width = 0.7,
     scaleBar = TRUE,
     ...)

## S3 method for class 'Synteny'
as.dist(m,
      diag = FALSE,
      upper = FALSE)

## S3 method for class 'Synteny'
print(x,
```

```
        quote = FALSE,
        right = TRUE,
        ...)
```

Arguments

x	An object of class Synteny.
bounds	Logical specifying whether to plot sequence boundaries as horizontal or vertical lines.
boxBlocks	Logical indicating whether to draw a rectangle around hits belonging to the same block of synteny.
colorBy	Numeric giving the index of a reference sequence, or a character string indicating to color by "neighbor", "frequency", or "none". (See details section below.)
colorRamp	A function that will return n colors when given a number n. Examples are rainbow, heat.colors, terrain.colors, cm.colors, or (the default) colorRampPalette.
barColor	Character string giving the background color of each bar.
barSides	Either a single logical determining whether to draw separators between bars, or two logicals indicating whether to draw inter-bar and intra-bar line separators, respectively.
horizontal	Logical indicating whether to plot the sequences horizontally (TRUE) or vertically (FALSE).
labels	Character vector providing names corresponding to each "identifier" for labels on the diagonal.
width	Numeric giving the fractional width of each bar between zero and one.
scaleBar	Logical controlling whether a scale bar is drawn when colorBy is "frequency". The scale bar displays the mapping between color and the level of sequence conservation. Not applicable when colorBy is a value other than "frequency".
gap	Distance between subplots, in margin lines.
line.main	If main is specified, line.main provides the line argument to mtext.
cex.labels	Magnification of the labels.
font.labels	Font of labels on the diagonal.
m	An object of class Synteny to be converted into a dist object.
diag	Logical determining whether to print the diagonal of the distance matrix.
upper	Logical determining whether to print the upper triangle of the distance matrix.
quote	Logical indicating whether to print the output surrounded by quotes.
right	Logical specifying whether to right align strings.
...	Other graphical parameters for pairs or plot, including: main, cex.main, font.main, and oma. Other arguments for print, including print.gap and max.

Details

Objects of class Synteny are stored as square matrices of list elements with dimnames giving the "identifier" of the corresponding sequences. The synteny matrix can be separated into three parts: along, above, and below the diagonal. Each list element along the diagonal contains an integer vector with the width of the sequence(s) belonging to that "identifier". List elements above the diagonal (column j > row i) each contain a matrix with "hits" corresponding to matches between sequences i and j. List elements below the diagonal each contain a matrix with "blocks" of synteny between sequences j and i.

The pairs method creates a scatterplot matrix from a Synteny object. Dot plots above the diagonal show hits between identifier i and j, where forward hits are colored in black, and hits to the reverse strand of identifier j are colored in red. Plots below the diagonal show blocks of synteny colored by their score, from green (highest scoring) to blue to magenta (lowest scoring).

The plot method displays a bar view of the sequences in the same order as the input object (x). The coloring scheme of each bar is determined by the colorBy argument, and the color palette is set by colorRamp. When colorBy is an index, the sequences are colored according to regions of shared homology with the specified reference sequence (by default 1). If colorBy is "neighbor" then shared syntenic blocks are connected between neighboring sequences. If colorBy is "frequency" then positions in each sequence are colored based on the degree of conservation with the other sequences. In each case, regions that have no correspondence in the other sequence(s) are colored barColor.

The as.dist method returns an object of class dist containing distances between each pair of sequences in a Synteny object. Distance is defined as one minus the hit coverage for the shorter of the two sequences in the pair.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

AlignSynteny, FindSynteny

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 # a small example:
 dbConn <- dbConnect(dbDriver("SQLite"), ":memory:")
 s1 <- DNAStringSet("ACTAGACCCAGACCGATAAACGGACTGGACAAG")
 s3 <- reverseComplement(s1)
 s2 <- c(s1, s3)
 Seqs2DB(c(c(c(s1, s2), s3),
         "XStringSet",
         dbConn,
         c("s1", "s2", "s2", "s3"))
 syn <- FindSynteny(dbConn, minScore=1)
 syn # Note:  > 100% hits because of sequence reuse across blocks
 pairs(syn, boxBlocks=TRUE)
 plot(syn)
 dbDisconnect(dbConn)
```

```
# a larger example:
db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
synteny <- FindSynteny(db, minScore=50)
class(synteny) # 'Synteny'
synteny

# accessing parts
i <- 1
j <- 2
synteny[i, i][[1]] # width of sequences in i
synteny[j, j][[1]] # width of sequences in j
head(synteny[i, j][[1]]) # hits between i & j
synteny[j, i][[1]] # blocks between i & j

# plotting
pairs(synteny) # dot plots
pairs(synteny, boxBlocks=TRUE) # boxes around blocks

plot(synteny) # bar view colored by position in genome 1
plot(synteny, barColor="#268FD6") # emphasize missing regions
plot(synteny, "frequency") # most regions are shared by all
plot(synteny, "frequency", colorRamp=rainbow) # change the colors
plot(synteny, "neighbor") # connect neighbors
}
```

Taxa *Taxa training and testing objects*

Description

Taxonomic classification is the process of assigning an organism a label that is part of a taxonomic hierarchy (e.g., Phylum, Class, Order, Family, Genus). Here, labels are assigned based on an organism's DNA or RNA sequence at a rank level determined by the classification's confidence. Class Taxa provides objects and functions for storing and viewing training and testing objects used in taxonomic classification.

Usage

```
## S3 method for class 'Taxa'
plot(x,
    y = NULL,
    showRanks = TRUE,
    n = NULL,
    ...)

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

```
## S3 method for class 'Taxa'
x[i, j, threshold]
```

Arguments

x	An object of class Taxa with subclass Train or Test.
y	An (optional) object of class Taxa with the opposite subclass as x.
showRanks	Logical specifying whether to show all rank levels when plotting an object of class Taxa and subclass Test. If TRUE (the default), then ranks are shown as (colored) concentric rings with radial lines delimiting taxa boundaries.
n	Numeric vector giving the frequency of each classification if x or y is an object of subclass Test, or the default (NULL) to treat all classifications as occurring once. Typically, specifying n is useful when the classifications represent varying numbers of observations, e.g., when only unique sequences were originally classified.
...	Other optional parameters.
i	Numeric or character vector of indices to extract from objects of class Taxa with subclass Test.
j	Numeric or character vector of rank levels to extract from objects of class Taxa with subclass Test.
threshold	Numeric specifying the confidence threshold at which to truncate the output taxonomic classifications. Note that threshold must be higher than the original for the classifications to change.

Details

Objects of class Taxa are stored as lists, and can have either subclass Train or Test. The function LearnTaxa returns an object of subclass Train, while the function IdTaxa can return an object of class Test.

Training objects are built from a set of reference sequences with known taxonomic classifications. List elements contain information required by IdTaxa for assigning a classification to test sequences.

Testing objects can be generated by IdTaxa from a Training object and a set of test sequences. Each list element contains the taxon, confidence, and (optionally) rank name of the taxonomic assignment.

The information stored in Taxa can be visualized with the plot function or displayed with print. Only objects of subclass Train can be subsetted without losing their class.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

LearnTaxa, IdTaxa

Run vignette("ClassifySequences", package = "DECIPHER") to see a related vignette.

Examples

```
data("TrainingSet_16S")
plot(TrainingSet_16S)

# import test sequences
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)

# remove any gaps in the sequences
dna <- RemoveGaps(dna)

# classify the test sequences
ids <- IdTaxa(dna, TrainingSet_16S, strand="top")
ids

plot(ids) # plot all rank levels
plot(ids[, 1:4]) # plot the first rank levels
plot(ids[j=c("rootrank", "class", "genus")]) # plot specific rank levels
plot(ids[threshold=70]) # plot high confidence classifications
```

TerminalChar *Determine the Number of Terminal Characters*

Description

Counts the number of terminal characters for every sequence in an `XStringSet`. Terminal characters are defined as a specific character repeated at the beginning and end of a sequence.

Usage

```
TerminalChar(myXStringSet,
             char = "")
```

Arguments

`myXStringSet`	An `XStringSet` object of sequences.
`char`	A single character giving the terminal character to count, or an empty character ("") indicating to count both gap ("-") and unknown (".") characters.

Value

A `matrix` containing the results for each sequence in its respective row. The first column contains the number of leading `char`, the second contains the number of trailing `char`, and the third contains the total number of characters in-between.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

IdLengths

Examples

```
fas <- system.file("extdata", "Bacteria_175seqs.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
t <- TerminalChar(dna)
head(t)
```

TileSeqs *Form a Set of Tiles for Each Group of Sequences.*

Description

Creates a set of k-mer tiles that represent each group of sequences in the database for downstream applications.

Usage

```
TileSeqs(dbFile,
         tblName = "Seqs",
         identifier = "",
         minLength = 26,
         maxLength = 27,
         maxTilePermutations = 10,
         minCoverage = 0.9,
         add2tbl = FALSE,
         processors = 1,
         verbose = TRUE,
         ...)
```

Arguments

dbFile	A database connection object or a character string specifying the path to a SQLite database file.
tblName	Character string specifying the table of sequences to use for forming tiles.
identifier	Optional character string used to narrow the search results to those matching a specific identifier. If "" then all identifiers are selected.
minLength	Integer providing the minimum number of nucleotides in each tile. Typically the same or slightly less than maxLength.
maxLength	Integer providing the maximum number of nucleotides in each tile. Tiles are designed primarily for this length, which should ideally be slightly greater than the maximum length of oligos used in downstream functions.
maxTilePermutations	Integer specifying the maximum number of tiles in each target site.

minCoverage	Numeric providing the fraction of coverage that is desired for each target site in the group. For example, a minCoverage of 0.9 request that additional tiles are added until 90% of the group is represented by the tile permutations.
add2tbl	Logical or a character string specifying the table name in which to add the result.
processors	The number of processors to use, or NULL to automatically detect and use all available processors.
verbose	Logical indicating whether to display progress.
...	Additional arguments to be passed directly to SearchDB.

Details

TileSeqs will create a set of overlapping tiles representing each target site in an alignment of sequences. The most common tile permutations are added until the desired minimum group coverage is obtained. The dbFile is assumed to contain DNAStringSet sequences (any U's are converted to T's).

Target sites with one more more tiles not meeting a set of requirements are marked with misprime equals TRUE. Requirements include minimum group coverage, minimum length, and maximum length. Additionally, tiles are required not to contain more than four runs of a single base or four di-nucleotide repeats.

Value

A data.frame with a row for each tile, and multiple columns of information. The row_names column gives the row number. The start, end, start_aligned, and end_aligned columns provide positioning of the tile in a consensus sequence formed from the group. The column misprime is a logical specifying whether the tile meets the specified constraints. The columns width and id indicate the tile's length and group of origin, respectively.

The coverage field gives the fraction of sequences containing the tile in the group that encompass the tile's start and end positions in the alignment, whereas groupCoverage contains the fraction of all sequences in the group containing a tile at their respective target site. For example, if only a single sequence out of 10 has information (no gap) in the first alignment position, then coverage would be 100% (1.0), while groupCoverage would be 10% (0.1).

The final column, target_site, provides the sequence of the tile.

Note

If add2tbl is TRUE then the tiles will be added to the database table that currently contains the sequences used for tiling. The added tiles may cause interference when querying a table of sequences. Therefore, it is recommended to add the tiles to their own table, for example, by using add2tbl="Tiles".

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

DesignPrimers

Examples

```
if (require("RSQLite", quietly=TRUE)) {
 db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
 tiles <- TileSeqs(db, identifier="Sphingomonadales")
}
```

TrainingSet_16S *Training Set for Classification of 16S rRNA Gene Sequences*

Description

A pre-trained classifier for 16S rRNA gene sequences generated by LearnTaxa.

Usage

```
data("TrainingSet_16S")
```

Format

A training set of class 'Taxa' * K-mer size: 8 * Number of rank levels: 10 * Total number of sequences: 2472 * Number of taxonomic groups: 2472 * Number of problem groups: 5 * Number of problem sequences: 8

Details

The original training sequences were pruned to a maximum of one sequence per group, as described in the 'Classifying Sequences' vignette.

Note

This 16S rRNA training set is provided for illustrative purposes only. It is highly recommended to use a more comprehensive training set when classifying real sequences. Examples of comprehensive training sets can be found at https://DECIPHER.codes/Downloads.html.

Source

Derived from version 16 of the RDP Training Set based on Bergey's Manual (Whitman *et al.*, 2012).

References

Whitman, W.B., Goodfellow, M., Kampfer, P., Busse, H.-J., Trujillo, M.E., Ludwig, W. & Suzuki, K.-i. (eds., 2012). Bergey's Manual of Systematic Bacteriology, 2nd ed., Springer-Verlag, New York, NY.

Examples

```
data(TrainingSet_16S)
TrainingSet_16S
plot(TrainingSet_16S)
```

Treeline *Construct a Phylogenetic Tree*

Description

Builds a phylogenetic tree from a set of sequences or distance matrix.

Usage

```
Treeline(myXStringSet = NULL,
         myDistMatrix = NULL,
         method = "ME",
         type = "dendrogram",
         cutoff = -Inf,
         showPlot = FALSE,
         standardDeviation = 0.2,
         fracRandomNNIs = 0.8,
         goalPercent = NA,
         tolerance = 1e-5,
         minIterations = 40,
         maxIterations = 400,
         maxTime = Inf,
         root = -1,
         collapse = -1,
         reconstruct = FALSE,
         costMatrix = NULL,
         model = MODELS,
         informationCriterion = "AICc",
         quadrature = FALSE,
         processors = 1,
         verbose = TRUE)
```

Arguments

myXStringSet	An AAStringSet, DNAStringSet, or RNAStringSet. Required if (a) not supplying myDistMatrix, (b) method is "ML", (c) method is "MP", or (d) reconstruct is not FALSE.
myDistMatrix	A symmetric N x N distance matrix with the values of dissimilarity between N sequences or an object of class 'dist'. If NULL (the default), myDistMatrix will be automatically determined from myXStringSet based on the distance between sequences using the DistanceMatrix function and the model (if specified). If not NULL, myDistMatrix is used to construct the initial tree, even when method is "ML" or "MP". If method is "ME", myDistMatrix determines the branch lengths and, therefore, overall tree length that is optimized. Missing values (i.e., NA) in myDistMatrix are imputed using the ultrametric method described by Makarenkov and Lapointe (2004).

`method` The phylogenetic method to be used. This should be one of `"ME"` for (balanced) minimum evolution (the default), `"ML"` for maximum likelihood, `"MP"` for maximum parsimony, `"NJ"` for neighbor joining, `"complete"` for complete-linkage, `"single"` for single-linkage, `"UPGMA"` for "unweighted pair group method with arithmetic mean", or `"WPGMA"` for "weighted pair group method with arithmetic mean". (See details section below.)

`type` Character string indicating the type of output desired. This should be one of `"dendrogram"` (the default), `"clusters"`, or `"both"`. (See value section below.)

`cutoff` A vector with the maximum edge length separating members of the same cluster. A negative value (the default) will prevent clustering. Multiple cutoffs may be provided in ascending or descending order. (See details section below.)

`showPlot` Logical specifying whether or not to plot the resulting dendrogram.

`standardDeviation`
Numeric determining the extent to which cophenetic distances are perturbed prior to constructing the starting candidate tree in each iteration. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`fracRandomNNIs` Numeric giving the fraction of stochastic nearest neighbor interchanges (NNIs) to perform when perturbing the starting candidate tree in each iteration after the first. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`goalPercent` Numeric providing the target percent change in score relative to the best observed tree after perturbing the starting tree in each iteration. If `goalPercent` is not `NA` (the default), `standardDeviation` and `fracRandomNNIs` are iteratively adjusted to reach `goalPercent` relative change in score. Reasonable non-default values are around `3` to `7`, as smaller values (e.g. `1`) will minimally explore tree space and larger values (e.g., `9`) may result in too much perturbation. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`tolerance` Numeric determining the relative convergence tolerance. Iterating will cease when the relative score has changed by less than `tolerance` for `minIterations` and has changed by less than `tolerance` per iteration on average. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`minIterations` Integer indicating the minimum number of iterations of optimization to perform. Lower values will result in faster convergence but may insufficiently explore tree space. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`maxIterations` Integer indicating the maximum number iterations of optimization to perform. More iterations will more thoroughly search tree space at the expense of added runtime. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`.

`maxTime` Numeric giving the maximum number of hours the algorithm is allowed to run before returning a result. Once `maxTime` is reached, the algorithm will proceed at the next available opportunity, even if `minIterations` is unmet. Only applicable if `method` is `"ME"`, `"ML"`, or `"MP"`. Note that setting a time limit may prevent reproducibility, even when setting a random number seed.

`root` One or more integers or character names specifying the indices of outgroup samples, `0` to midpoint root the dendrogram, or `-1` (the default) to perform minimum variance rooting (Mai *et al.*, 2017). If more than one outgroup index or name is provided, the `root` will be determined such that the entire outgroup is below the `root`. Partial name matching is supported so long as the matches are unambiguous.

`collapse` Numeric controlling which internal edges of the tree are removed by collapsing their nodes. If `collapse` is zero then nodes at the same height will be collapsed to a single node, resulting in a multifurcating tree. When `collapse` is greater than zero, nodes that are within `collapse` difference in height are made into a single node. A value of `collapse` less than zero (the default) will ensure that the `dendrogram` is purely bifurcating. Note that `collapse` has no effect on cluster numbers or `cutoff`.

`reconstruct` Logical or numeric determining whether to perform ancestral state reconstruction when `myXStringSet` is specified. If `TRUE`, ancestral character states are determined at internal nodes of the `dendrogram` and provided as a `"state"` attribute. Ancestral states are determined as the most parsimonious state according to the `"costMatrix"`, unless `method` is `"ML"`, in which case ancestral states are determined as those with the highest (marginal) likelihood. A numeric value between zero and one (exclusive) can be provided when `method` is `"ML"` to require a specific fraction of the state's likelihood to be greater than that of all alternative states, otherwise a more ambiguous degeneracy code is used. Only applicable if `type` is `"dendrogram"` (the default) or `"both"`.

`costMatrix` Either `NULL` (the default) or a symmetric matrix setting the penalties used in Sankoff parsimony. The default (`NULL`) will apply a uniform cost of `1` for standard character state changes and `0` otherwise (i.e., equivalent to Fitch parsimony). If a `matrix` then the states are taken from its row or column names. Only applicable if `method` is `"MP"`, or `reconstruct` is `TRUE` and `method` is not `"ML"`.

`model` One or more of the available `MODELS` of evolution provided as a character vector or list with components 'Protein' and/or 'Nucleotide' if `method` is `"ML"`. Automatic model selection based on the `informationCriterion` will be performed for `"ML"` if more than one `model` is provided. For `"ME"`, if `myDistMatrix` is `NULL`, a single model of evolution can be specified for generating the distance matrix from `myXStringSet`.

`informationCriterion`
Character string specifying which information criterion to use in automatic model selection when `method` is `"ML"`. Must be either `"AICc"` or `"BIC"`. The best `model` is automatically chosen based on the `informationCriterion` calculated from the likelihood and the sample size (defined as the average number of sites for sequences in `myXStringSet`). Only applicable if `method` is `"ML"`.

`quadrature` Logical determining whether to use the Laguerre quadrature or equal-sized bins when discretizing the rate distribution across sites when `method` is `"ML"`. The default (`FALSE`) is to use equal-sized bins for direct comparison among likelihoods computed by other programs, although the Laguerre quadrature offers a modest improvement in accuracy for the same number of rates (Felsenstein, 2001).

`processors` The number of processors to use, or `NULL` to automatically detect and use all available processors. Note, the number of processors in some steps is automatically selected between `1` and `processors` to optimize performance, and specifying a number for `processors` does not guarantee that number will be used throughout.

`verbose` Logical indicating whether to display progress.

Details

Treeline builds a phylogenetic tree using either myXStringSet and/or myDistMatrix. The output is either a dendrogram and/or data.frame containing cluster numbers. The Treeline algorithm is described in Wright (2026).

Multiple methods of tree building are supported:

(1) Minimum evolution: ME iteratively minimizes the (balanced) tree length according to a distance matrix, as described by Pauplin (2000). According to Gonnet (2012) and Spirin *et al.* (2024), the ME criterion performs best overall on benchmarks constructed from real genes and is therefore the default method. ME is also the fastest of the optimized methods. Branch lengths are determined from myDistMatrix or, if missing (NULL), calculated from the length-normalized Hamming distances between sequences in myXStringSet unless a model is specified.

(2) Maximum likelihood: ML iteratively maximizes the likelihood of the tree and any free model parameters with respect to the aligned sequences (myXStringSet). One or more MODELS of sequence evolution must be specified, of which the best model is automatically selected based on the informationCriterion. The ML criterion performs well on benchmarks constructed from real genes when given sufficiently long and variable alignments. Branch lengths are in units of substitution per site.

(3) Maximum parsimony: MP iteratively maximizes the parsimony of a tree given aligned sequences (myXStringSet) and a costMatrix. The default cost matrix is binary, corresponding to Fitch (1971) parsimony. The costMatrix often has a large influence over the result, and more biologically reasonable cost matrices will likely improve the resulting tree. See the examples below for non-uniform cost matrices. Here, branch lengths are in units of the average cost per site, which is equivalent to averages changes per site for a binary cost matrix.

(4) Neighbor-joining: NJ uses the Neighbor-Joining method proposed by Saitou and Nei (1987), which creates an approximate minimum evolution tree from a distance matrix (myDistMatrix). The NJ criterion is a greedy heuristic approach to minimum evolution, and can be considered a less accurate alternative to ME. Notably, NJ is faster than ME for moderate numbers of sequences, since it only computes a single tree, but is potentially slower than ME for large numbers of sequences due to its cubic time complexity.

(5) Ultrametric: The method complete assigns clusters using complete-linkage so that members of the same cluster are no more than cutoff distance apart. The method single assigns clusters using single-linkage so that members of the same cluster are within cutoff of at least one other member. UPGMA and WPGMA assign clusters using average-linkage which is a compromise between the sensitivity of complete-linkage clustering to outliers and the tendency of single-linkage clustering to connect distant members. UPGMA produces an unweighted tree, where each leaf contributes equally to the average edge lengths, whereas WPGMA produces a weighted result.

For methods "ME", "ML", and "MP", candidate trees are iteratively optimized through rounds of nearest neighbor interchanges ("climbs") followed by testing replacement of any remaining subtree differences on the best observed tree ("grafts"). Candidate trees are generated with a heuristic variant of neighbor joining after perturbing the best observed tree's cophenetic distance matrix, followed by stochastic NNIs if fracRandomNNIs is greater than 0. The value of fracRandomNNIs is adaptively varied to reach goalPercent relative score for each iteration's initial (perturbed) tree, unless goalPercent is NA (the default). This process results in gradual improvement until reaching a best tree, which is returned if insufficient score improvement is made for minIterations, unless maxIterations or maxTime is reached.

The objective of optimization is to find the global optimum, but it is necessary to set a random seed for exact reproducibility since optimization is stochastic and may only find a local optimum. Setting `maxTime` may prevent reproducibility if iteration terminates prior to reaching convergence or `maxIterations`. Also, not all math operations (e.g., logarithm) have standardized implementations across platforms, so reproducibility is not guaranteed on different machines. The results are not necessarily significant even if they are reproducible, and it possible to use bootstrapping to gauge support for different partitions. When `method` is `"ML"`, aBayes support values are provided that are a reasonable representation of branch support.

The returned dendrogram has information stored in its attributes, which can be accessed with the `attributes` or `attr` functions. For maximum likelihood trees, the edges have a `"probability"` attribute representing the aBayes support probability (Anisimova, 2011). If `reconstruct` is not `FALSE`, each edge of the tree will have a `"state"` attribute representing the node's predicted ancestral state. Also, when `reconstruct` is not `FALSE`, maximum likelihood trees will provide the likelihoods at each site and other `methods` will provide a state transition matrix based on parsimony.

When non-negative `cutoff`(s) are supplied, `Treeline` will assign clusters based on edge lengths in the tree. Multiple cutoffs can be provided in sorted order. If the `cutoffs` are provided in *descending* order then clustering at each new value of `cutoff` is continued within the prior `cutoff`'s clusters. In this way clusters at lower values of `cutoff` are completely contained within their umbrella clusters at higher values of `cutoff`. This is useful for defining taxonomy, where groups need to be hierarchically nested. If multiple cutoffs are provided in *ascending* order then clustering at each level of `cutoff` is independent of the prior level.

Value

If `type` is `"dendrogram"` (the default), then a tree of class `dendrogram` is returned with attributes including pertinent information. If `type` is `"clusters"`, then a `data.frame` is returned with dimensions $N * M$, where each one of N sequences is assigned to a cluster at the M-level of `cutoff`. If `type` is `"both"` then a list is returned containing both the `"clusters"` and `"dendrogram"` outputs.

Note

Cophenetic distance between leaves of the `dendrogram` is often defined as the sum of branch lengths separating the leaves, also known as the patristic distance. This is the typical phylogenetic interpretation but different than that for dendrograms produced by `hclust` where leaves are merged at a height defined by `hclust` as their cophenetic distance. Hence, always use `Cophenetic` (rather than `cophenetic`) to compute patristic distances where the length between leaves is desired.

Negative branch lengths are possible for `ME` or `NJ` trees when the data violates the triangle inequality. Since negative branch lengths are artifactual, they are set to zero when encountered. This can cause the sum of branch lengths to differ from the overall (balanced) tree length that represents the score optimized by the `ME method`. However, the score will remain consistent with other `ME` programs' calculations for the same tree topology. Similarly, branch lengths are set to a very small (non-zero) value for the `ML method` due to machine precision.

Ambiguity codes (e.g., "N" or "X") are handled differently across `methods`. The `MP method` will consider them if part of the `costMatrix`. In contrast, the `ML method` splits the likelihood of ambiguities across the character states they represent. All other `methods` depend on how the distance matrix is built for the treatment of ambiguities (see `DistanceMatrix`). Relatedly, the treatment of gaps depends on the `model` of evolution that is used.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Anisimova M., *et al.* (2011) Survey of branch support methods demonstrates accuracy, power, and robustness of fast likelihood-based approximation schemes. *Syst Biol.*, **60(5)**, 685-99.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17(6)**, 368-376.

Felsenstein J. (2001) Taking variation of evolutionary rates between sites into account in inferring phylogenies. *Journal of molecular evolution*, **53**, 447-455.

Fitch, W. M. (1971) Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, **20**:406-416.

Gonnet, G. H. (2012) Surprising results on phylogenetic tree building methods based on molecular sequences. *BMC Bioinformatics*, **13**, 148.

Mai, U., *et al.* (2017) Minimum variance rooting of phylogenetic trees and implications for species tree reconstruction. *PLoS ONE*, **12(8)**, e0182238.

Makarenkov V., and Lapointe, F. (2004) A weighted least-squares approach for inferring phylogenies from incomplete distance matrices. *Bioinformatics*, **20(13)**, 2113-2121.

Pauplin, Y. (2000) Direct Calculation of a Tree Length Using a Distance Matrix. *J Mol Evol*, **51(1)**, 41-47.

Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, **4(4)**, 406-425.

Sankoff, D. (1975) Minimal mutation trees of sequences. *SIAM Journal of Applied Math*, **28**.

Spirin, S., *et al.* (2024) PhyloBench: A Benchmark for Evaluating Phylogenetic Programs. *Molecular Biology and Evolution*, **41(6)**, 1-11.

Wright, E. (2026) Treeline Provides a Unified Strategy for Optimising Phylogenetic Trees Under Alternative Criteria. *Molecular Ecology Resources*, **26(5)**, e70163.

See Also

`Clusterize`, `Cophenetic`, `DistanceMatrix`, `MapCharacters`, `MODELS`, `ReadDendrogram`, `WriteDendrogram`, `Zipline`

Run `vignette("GrowingTrees", package = "DECIPHER")` to see a related vignette.

Examples

```
# using the matrix from Saitou and Nei (1987)
m <- matrix(0, 8, 8) # only the lower triangle is used
m[2:8, 1] <- c(7, 8, 11, 13, 16, 13, 17)
m[3:8, 2] <- c(5, 8, 10, 13, 10, 14)
m[4:8, 3] <- c(5, 7, 10, 7, 11)
m[5:8, 4] <- c(8, 11, 8, 12)
m[6:8, 5] <- c(5, 6, 10)
m[7:8, 6] <- c(9, 13)
```

```
m[8, 7] <- 8

# returns an object of class "dendrogram"
tree <- Treeline(myDistMatrix=m, cutoff=10, method="NJ", showPlot=TRUE)

# example of specifying multiple cutoffs
clusters <- Treeline(myDistMatrix=m, method="UPGMA", type="clusters", cutoff=c(2,6,10,20))
head(clusters)

# example of creating a complete-linkage tree from an alignment
fas <- system.file("extdata", "50S_ribosomal_protein_L2.fas", package="DECIPHER")
dna <- readDNAStringSet(fas)
alignments <- AlignTranslation(dna, verbose=FALSE, type="both")
dna <- alignments[[1]]
aa <- alignments[[2]]
dna # input alignment
d <- DistanceMatrix(dna, type="dist") # returns an object of class 'dist'
complete <- Treeline(myDistMatrix=d, method="complete", cutoff=0.05, showPlot=TRUE)

# example of minimum evolution (ME) tree optimization (the default)
treeME <- Treeline(dna, processors=1) # the recommended way to build trees
plot(treeME, leaflab="none")

# compare the distance matrix to the cophenetic (patristic) distance matrix
plot(d, Cophenetic(treeME),
 xlab="Pairwise distance", ylab="Patristic distance",
 asp=1, pch=46, col="#00000033")
abline(a=0, b=1)

# example of minimum evolution (ME) with a model of evolution
treeME_DNA <- Treeline(dna, model="TN93+F", processors=1L) # for DNA
treeME_AA <- Treeline(aa, model="WAG+F", processors=1L) # for AA
plot(Cophenetic(treeME_DNA), Cophenetic(treeME_AA),
 xlab="DNA patristic distance", ylab="AA patristic distance",
 asp=1, pch=46, col="#00000033")
abline(a=0, b=1)

# example of maximum parsimony (MP) tree optimization
costs <- matrix(c(0, 2, 1, 2, 2, 0, 2, 1, 1, 2, 0, 2, 2, 1, 2, 0), 4)
dimnames(costs) <- list(DNA_BASES, DNA_BASES)
costs # the cost matrix
treeMP <- Treeline(dna, method="MP", reconstruct=TRUE, costMatrix=costs, maxTime=0.001)

# display ancestral states on each edge
start <- 50 # starting position in alignment
end <- 52 # ending position in alignment
tree <- dendrapply(treeMP,
 function(x) {
  attr(x, "edgetext") <- substring(attr(x, "state"),
   start,
   end)
  x
 })
```

```
plot(tree,
 edgePar=list(p.col=NA,
  p.border=NA,
  t.col=c("#AA3355", "#33FFFF")),
 edge.root=FALSE,
 leaflab="none")

# example of maximum likelihood tree optimization
treeML <- Treeline(head(dna, 10), method="ML", reconstruct=TRUE, maxTime=0.001)

# example of accessing and using the attributes
attributes(treeML) # show all attributes at a node
plot(dendrapply(treeML,
 function(x) {
  s <- attr(x, "probability")
  if (!is.null(s))
   attr(x, "edgetext") <- formatC(as.numeric(s), digits=2, format="f")
  attr(x, "edgePar") <- list(p.col=NA, p.border=NA, t.col="#CC55AA", t.cex=0.7)
  x
 }),
 horiz=TRUE)

# construct a tanglegram comparing two trees back-to-back
layout(matrix(1:2, ncol=2))
tree1 <- treeME # tree on left
tree2 <- treeMP # tree on right
tree1 <- reorder(tree1, unlist(tree1))
tree2 <- reorder(tree2, unlist(tree2))
layout(matrix(1:2, nrow=1))
par(mai=c(0.5, 0, 0.5, 0.5)) # add space on right
plot(tree1,
 main="First tree (ME)",
 horiz=TRUE, leaflab="none")
par(mai=c(0.5, 0.5, 0.5, 0)) # add space on left
plot(tree2,
 main="Second tree (MP)",
 horiz=TRUE, leaflab="none",
 xlim=c(0, attr(tree2, "height")))
segments(par("usr")[1] - 1.05*diff(grconvertX(0:1, 'inches', 'user')),
 match(labels(tree2), labels(tree1)),
 -0.05*diff(grconvertX(0:1, 'inches', 'user')),
 seq_len(attr(tree2, "members")),
 xpd=NA)

# example of supplying an amino acid cost matrix for MP
codons <- getGeneticCode("1") # the standard genetic code
codons <- tapply(names(codons), codons, c)
costs <- outer(codons,
 codons,
 function(x, y)
  mapply(function(x, y) mean(adist(x, y)), x, y))
diag(costs) <- 0 # zero cost for no change
costs # minimum number of nucleotide changes to switch amino acids
```

```
treeMP_AA1 <- Treeline(aa, method="MP", costMatrix=costs, maxTime=0.001)
plot(Cophenetic(treeMP_AA1), Cophenetic(treeMP))

# alternative approach to obtaining an amino acid cost matrix
data(BLOSUM)
subM <- BLOSUM[AA_STANDARD, AA_STANDARD, "62"] # BLOSUM62 matrix
subM <- diag(subM) - subM # standardize to diagonal = 0
subM <- (subM + t(subM))/2 # make symmetric
treeMP_AA2 <- Treeline(aa, method="MP", costMatrix=subM, maxTime=0.001)
plot(Cophenetic(treeMP_AA1), Cophenetic(treeMP_AA2)) # different lengths
```

`TrimDNA` *Trims DNA Sequences to the High Quality Region Between Patterns*

Description

Aids in trimming DNA sequences to the high quality region between a set of patterns (e.g., primers) that are potentially present on the left and right sides.

Usage

```
TrimDNA(myDNAStringSet,
        leftPatterns,
        rightPatterns,
        type = "ranges",
        quality = NULL,
        maxDistance = 0.1,
        minOverlap = 5,
        allowInternal = TRUE,
        alpha = 0.2,
        threshold = 0.01,
        maxAverageError = 0.02,
        maxAmbiguities = 0,
        minWidth = 36,
        verbose = TRUE)
```

Arguments

`myDNAStringSet`	A `DNAStringSet` or `QualityScaledDNAStringSet` object containing the sequences to be trimmed. If `"type"` is `"sequences"` then the output class will match the class of `myDNAStringSet`.
`leftPatterns`	A `DNAStringSet` or character vector of patterns to remove from the left side of `myDNAStringSet`, or `""` to prevent trimming patterns on the left.
`rightPatterns`	A `DNAStringSet` or character vector of patterns to remove from the right side of `myDNAStringSet`, or `""` to prevent trimming patterns on the right.
`type`	Character string indicating the type of results desired. This should be either `"ranges"`, `"sequences"` or `"both"`.

quality — Either NULL or a PhredQuality, SolexaQuality, or IlluminaQuality object containing the quality scores corresponding to myDNAStringSet. By default, quality is inherited from the input if myDNAStringSet is a QualityScaledDNAStringSet, otherwise quality trimming is skipped if quality is NULL (the default).

maxDistance — Numeric between zero and one giving the maximum distance of a match from the leftPatterns and rightPatterns to initiate trimming. For example, 0.1 (the default) would allow up to 10% mismatches between a pattern and sequence.

minOverlap — Integer specifying the minimum number of nucleotides the leftPatterns and rightPatterns must overlap a sequence to initiate trimming.

allowInternal — Logical initiating whether to search for the leftPatterns and rightPatterns within myDNAStringSet, or (FALSE for) only overlapping the ends.

alpha — Numeric between zero and one giving the smoothing parameter for an exponential moving average that is applied to the quality scores before trimming. Higher values result in less smoothing than lower values.

threshold — Numeric between zero and one specifying the threshold above which to trim the poor quality regions of the sequence. Higher values allow more sequence to be preserved at the expense of a greater error rate.

maxAverageError — Numeric between zero and threshold indicating the maximum average error rate of the trimmed region of the sequence. Trimmed sequences with average error rates above maxAverageError will be rejected. Note that the expected number of errors in a sequence is equal to the average error rate multiplied by the length of the sequence.

maxAmbiguities — Numeric between zero and one giving the maximum fraction of ambiguous (e.g., "N") positions that are tolerated within the trimmed region of the sequence. Trimmed sequences with a greater fraction of ambiguities than maxAmbiguities will be rejected.

minWidth — Integer giving the minimum number of nucleotides a pattern must overlap the sequence to initiate trimming.

verbose — Logical indicating whether to display progress.

Details

After a sequencing run, it is often necessary to trim the resulting sequences to the high quality region located between a set of patterns. TrimDNA works as follows: first left and right patterns are identified within the sequences if allowInternal is TRUE (the default). If the patterns are not found internally, then a search is conducted at the flanking ends for patterns that partially overlap the sequence. The region between the leftPatterns and rightPatterns is then returned, unless quality information is provided. Note that the patterns must be in the same orientation as the sequence, which may require using the reverseComplement of a PCR primer.

If quality scores are provided, these are converted to error probabilities and an exponential moving average is applied to smooth the signal in log-space. The longest region between the leftPatterns and rightPatterns where the average error probability is below threshold is then returned, so long as it has an average error rate of at most maxAverageError. Default error cutoffs were set to maximize sensitivity and specificity across diverse read types and genomes, but it is advisable to tune input parameters to the intended application.

Value

TrimDNA can return two types of results: IRanges that can be used for trimming myDNAStringSet, or a trimmed DNAStringSet or QualityScaledDNAStringSet containing only those sequences over minWidth nucleotides after trimming. Note that ambiguity codes (IUPAC_CODE_MAP) are supported in the leftPatterns and rightPatterns, but not in myDNAStringSet to prevent trivial matches (e.g., runs of N's).

If type is "ranges" (the default) the output is an IRanges object with the start, end, and width of every sequence. This information can be accessed with the corresponding accessor function (see examples below). Note that the start will be 1 and the end will be 0 for sequences that were not at least minWidth nucleotides after trimming.

If type is "sequences" then the trimmed sequences are returned that are at least minWidth nucleotides in length.

If type is "both" the output is a list of two components, the first containing the ranges and the second containing the sequences.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

CorrectFrameshifts

Examples

```
# simple example of trimming a single sequence
dna <- DNAStringSet("AAAAAAAAAATTACTTCCCCCCCCCC")
qscores <- PhredQuality("0000000000AAAAAAAAAAAAAAAA")

x <- TrimDNA(dna,
            leftPatterns="AAAAAA",
            rightPatterns="CCCCCC",
            quality=qscores,
            minWidth=1,
            allowInternal=TRUE,
            type="both")

x[[1]]
start(x[[1]])
end(x[[1]])
width(x[[1]])
subseq(dna, start(x[[1]]), end(x[[1]]))
x[[2]]

# example of trimming a FASTQ file by quality scores
fpath <- system.file("extdata",
 "s_1_sequence.txt",
 package="Biostrings")
reads <- readQualityScaledDNAStringSet(fpath)
trimmed <- TrimDNA(reads,
```

```
 leftPatterns="",
 rightPatterns="",
 type="sequences")
trimmed
DNAStringSet(trimmed) # drop the qualities

# merge and trim paired-end reads
faq1 <- system.file("extdata", "Simulated_Illumina_Read1.fq.gz", package="DECIPHER")
faq2 <- system.file("extdata", "Simulated_Illumina_Read2.fq.gz", package="DECIPHER")
read1 <- readQualityScaledDNAStringSet(faq1)
read2 <- readQualityScaledDNAStringSet(faq2)
read1
read2
merge <- AlignPairs(read1,
 reverseComplement(read2),
 type="sequences",
 bandWidth=200)
merge
merge <- TrimDNA(merge$Consensus,
 DNAStringSet("GTGYCAGCMGCCGCGGTAA"), # forward primer
 reverseComplement(DNAStringSet("GGACTACNVGGGTWTCTAAT")), # reverse primer
 type="sequences")
merge
clus <- Clusterize(RemoveGaps(merge), cutoff=0.01, singleLinkage=TRUE)
sort(table(clus)) # ideal result is 20 clusters of 100 sequences
```

`WriteDendrogram` *Write a Dendrogram to Newick Format*

Description

Writes a dendrogram object to a file in Newick (also known as New Hampshire) parenthetic format.

Usage

```
WriteDendrogram(x,
                file = "",
                quote = "'",
                space = " ",
                internalLabels = TRUE,
                digits = 10,
                append = FALSE,
                unroot = FALSE)
```

Arguments

`x`	An object of class `dendrogram` or a list object containing `dendrograms`.
`file`	A connection or a character string naming the file path where the tree should be exported. If "" (the default), the tree is printed to the standard output connection, typically the console.

quote	A single character used to quote labels, or an empty character string (i.e., "") to avoid quoting labels.
space	A single character (e.g., "_") used to replace spaces in labels, or a space (i.e., " ") to leave spaces intact.
internalLabels	Logical indicating whether to write any "edgetext" preceding a node as an internal node label.
digits	The maximum number of significant digits to print for edge lengths.
append	Logical indicating whether to append to an existing file. Only applicable if file is a character string. If FALSE (the default), then the file is overwritten.
unroot	Logical determining whether to unroot x if possible.

Details

WriteDendrogram will write a dendrogram object to a file in standard Newick format. Note that special characters (commas, square brackets, colons, semi-colons, and parentheses) present in leaf labels will likely cause a broken Newick file unless quote is a single (the default) or double quotation mark.

Value

NULL.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

References

Wright, E. (2026) Treeline Provides a Unified Strategy for Optimising Phylogenetic Trees Under Alternative Criteria. *Molecular Ecology Resources*, **26(5)**, e70163.

See Also

Treeline, ReadDendrogram

Run vignette("GrowingTrees", package = "DECIPHER") to see a related vignette.

Examples

```
dists <- matrix(c(0, 10, 20, 10, 0, 5, 20, 5, 0),
    nrow=3,
    dimnames=list(c("dog", "elephant", "horse")))
dend <- Treeline(myDistMatrix=dists, method="NJ")
# write one dendrogram
WriteDendrogram(dend)
# write multiple dendrograms
WriteDendrogram(list(dend, dend))
# write an unrooted dendrogram
WriteDendrogram(dend, unroot=TRUE)
# unquote leaf labels
WriteDendrogram(dend, quote="")
```

WriteGenes *Write Genes to a File*

Description

Writes predicted genes to a file in GenBank (gbk) or general feature format (gff).

Usage

```
WriteGenes(x,
           file = "",
           format = "gbk",
           append = FALSE)
```

Arguments

x	An object of class Genes.
file	A connection or a character string naming the file path where the tree should be exported. If "" (the default), the tree is printed to the standard output connection, the console unless redirected by sink.
format	Character specifying "gbk" or "gff" output format.
append	Logical indicating whether to append to an existing file. Only applicable if file is a character string. If FALSE (the default), then the file is overwritten.

Details

WriteGenes will write a "Genes" object to a GenBank (if format is "gbk") or general feature format (if format is "gff") file.

Value

NULL.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

ExtractGenes, FindGenes, Genes-class

Examples

```
# import a test genome
fas <- system.file("extdata",
 "Chlamydia_trachomatis_NC_000117.fas.gz",
 package="DECIPHER")
genome <- readDNAStringSet(fas)

x <- FindGenes(genome)
WriteGenes(x[1:10,], format="gbk")
WriteGenes(x[1:10,], format="gff")
```

`Zipline` *Summarize Multiple Phylogenetic Trees*

Description

Generates a summary dendrogram or distance matrix from a set of phylogenetic trees using linear regression of transformed distances.

Usage

```
Zipline(x,
        type = "dendrogram",
        distance = "length",
        weights = 1,
        bootstraps = 0,
        power = 0.5,
        concordanceVectors = FALSE,
        verbose = TRUE,
        ...)
```

Arguments

`x`	A list containing one `dendrogram` object per list element.
`type`	Character string indicating the type of output desired. This should be one of `"dendrogram"` (the default), `"dist"`, or `"both"`. (See value section below.)
`distance`	The type of distance measure(s) to use. This should be `"length"` (the default), `"edges"`, and/or named edge attributes. (See the `Cophenetic` manual page for more information.)
`weights`	Either a single number (the default) or a numeric vector with one weight corresponding to each `dendrogram` in `x`. Specifying `weights` as proportional to trustworthiness (e.g., number of aligned sites originally used to construct each dendrogram) may modestly improve summary tree accuracy.
`bootstraps`	The number of bootstrap replicates desired or zero (the default) for no replicates. Bootstrapping entails resampling (with replacement) the set of trees in `x` and recording the fraction of times an edge partitioning the same sets of leaves was observed across replicates.

power — A single number determining the transformed space in which regression is performed. The default (0.5) will perform regression in square root space, while setting power to 1 would perform traditional regression. Smaller values will diminish the influence of large outliers at the expense of losing some of their information.

concordanceVectors — A logical determining whether to calculate concordance vectors, a numeric vector with a weight for each dendrogram object (list element) in x, or a character string specifying an attribute name to use as the weight of each internal branch in x. Only applicable when type is not "dist". (See details section below.)

verbose — Logical indicating whether to display progress.

... — Additional arguments to be passed directly to Treeline for generating the output dendrogram.

Details

The genealogies of gene trees are often discordant with each other and the species tree due to biological processes and technical difficulties. Given sufficient gene trees, it is feasible to reconstruct the species tree that best summarizes all gene trees. This is the purpose of Zipline, which is named for its objective of zippy (fast) and accurate summary tree inference via linear regression to connect different trees.

Zipline works by first regressing gene tree distances onto the average inter-leaf distances to calculate the relative rate of evolution for each gene tree. Next, the function regresses inter-leaf distances onto the per tree rate to derive the best estimate of distances between leaves. By default, this distance matrix is used to construct a summary tree with Treeline.

Regressions are performed in square root space (by default) to mitigate the effect of outliers on estimated slopes. The use of regression makes the Zipline algorithm robust to error and effortlessly capable of handling missing data (e.g., absent genes). Zipline also handles duplicated labels in the same tree (e.g., multi-copy genes) by collapsing to the minimum distance. In this manner, all input trees do not need to have the same set of (leaf) labels. Regression also permits the incorporation of weights to bias to the distance estimate to towards more trustworthy trees in x.

Multiple measures of distance are supported. Historically the interedge distance ("edges") was commonly used for species tree inference, but recently path length ("length") and other measures have been shown to rival interedge distance. Dendrograms that include branch supports (e.g., "edgetext") may offer higher accuracy and make hybrid distance measures possible. (See examples section below.)

Output distances (and edge lengths) are in the same units as the distance measure. That is, if "length" is specified then output distances are in the units of branch length (e.g., substitutions per site). Other distance measures, such as the sum of edge support values, are less easily interpretable although they remain in the expected units. Note that all trees in x are expected to share the same units.

Bootstrapping is incorporated at the level of sampling gene trees within x (i.e., multi-locus bootstrapping). As with most bootstrap procedures, this requires a large enough sample set (i.e., x) to adequately represent the data distribution. Bootstrap supports (as fractions) are provided via "edgetext" attributes on internal branches of the tree.

If concordanceVectors is TRUE, then gene concordance vectors are provided as "vector" attributes on internal branches of the output dendrogram. Concordance vectors have seven named

integers: the number of input dendrograms with concordant ("CF") versus discordant ("DF1" and "DF2") topologies for both splits (bipartitions) and quartets, and the total number of "decisive" topologies. The total counts of concordant plus discordant topologies will be less than or equal to the number of decisive topologies, defined as input trees with at least one taxa present from each of the four clades immediately surrounding a branch.

It is also possible for concordanceVectors to be a vector with weights for each input tree. This is useful to weigh the concordance vector by the overall confidence (e.g., average support) or information content (e.g., number of informative sites) underlying each tree. Alternatively, concordanceVectors can be the name of an attribute to use as the weight of each branch in the input trees. For example, it is feasible to specify a character string corresponding to branch support values (e.g., "edgetext"), which would result in concordance vector quantities representing the sum of support. When specifying a character string, the number of decisive trees will remain unchanged, since this value does not require correspondence to a specific branch.

Value

If type is "dendrogram" (the default), then a summary tree of class dendrogram is returned with "edgetext" attributes if bootstraps is positive. If type is "dist", then a summary distance matrix of class dist is returned. If type is "both" then a list is returned containing both the "dendrogram" and "dist" outputs.

Author(s)

Erik Wright <Erik.Wright@bcm.edu>

See Also

Cophenetic, ReadDendrogram, Treeline, WriteDendrogram

Examples

```
# import a set of single-copy gene trees from plant genomes
newick <- system.file("extdata", "Plant_gene_trees.newick.xz", package="DECIPHER")
trees <- ReadDendrogram(newick)
length(trees) # number of dendrograms
plot(trees[[1]], main="First gene tree")

# using the default distance (path length)
species_tree <- Zipline(trees)
plot(species_tree, main="Species tree under default settings")

# using the interedge distance
species_tree <- Zipline(trees, distance="edges")
plot(species_tree, main="Species tree from interedge distances")

# normalizing branch supports from 0% to 100%
trees <- lapply(trees,
 dendrapply,
 function(x) {
  b <- as.numeric(attr(x, "edgetext"))
  if (length(b) == 0) { # leaf
```

```
   attr(x, "support") <- 1
  } else {
   attr(x, "support") <- b/100
  }
  x
 })

# using branch supports as distance
species_tree <- Zipline(trees, distance="support")
plot(species_tree, main="Species tree from branch supports")

# using a hybrid of branch confidence times length
species_tree <- Zipline(trees, distance=c("support", "length"))
plot(species_tree, main="Species tree from branch length*support")

## Not run:
# calculating concordance vectors based on counts
species_tree <- Zipline(trees, concordanceVectors=TRUE)
attr(species_tree, "vector") # concordance vector at the top branch

# calculating concordance vectors based on branch support
species_tree <- Zipline(trees, concordanceVectors="support")
attr(species_tree, "vector") # concordance vector at the top branch

## End(Not run)
```

Index

∗ **datasets**
BLOSUM, 32
deltaGrules, 66
deltaGrulesRNA, 67
deltaHrules, 68
deltaHrulesRNA, 69
deltaSrules, 70
deltaSrulesRNA, 71
HEC_MI, 112
MIQS, 145
MMLSUM, 146
NonCodingRNA, 153
PAM, 156
PFASUM, 157
RESTRICTION_ENZYMES, 167
TrainingSet_16S, 187
∗ **data**
AA_REDUCED, 7
MODELS, 147
∗ **package**
DECIPHER-package, 4
[.Genes *(*Genes*)*, 110
[.Synteny *(*Synteny*)*, 179
[.Taxa *(*Taxa*)*, 182

AA_REDUCED, 7, *48*
Add2DB, 7, *101*, *117*
AdjustAlignment, 9, *25*, *178*
AlignDB, 11, *21*, *25*, *28*
AlignPairs, 14, *122*, *174*
AlignProfiles, *10*, *13*, *17*, 18, *24–26*, *28*
AlignSeqs, *11*, *13*, *21*, 22, *27*, *28*, *142*, *166*, *169*, *178*
AlignSynteny, *21*, 25, *25*, *28*, *87*, *108*, *181*
AlignTranslation, *11*, *13*, *21*, *25*, 27, *58*
AmplifyDNA, 29, *44*, *77*, *84*, *144*
Array2Matrix, 31, *74*, *152*
as.dist.Synteny *(*Synteny*)*, 179

BLOSUM, 32
BrowseDB, *8*, 33, *37*, *177*
BrowseSeqs, *34*, 35

c.Taxa *(*Taxa*)*, 182
CalculateEfficiencyArray, 39
CalculateEfficiencyFISH, 41, *80*
CalculateEfficiencyPCR, *30*, *31*, 42, *77*, *84*, *144*
Clusterize, 44, *193*
Codec, *13*, 49, *176*
ConsensusSequence, *17*, *37*, 51, *91*, *113*
Cophenetic, 53, *193*, *202*, *204*
CorrectFrameshifts, *28*, 55, *155*, *198*
CreateChimeras, 59, *101*

DateTree, 60
DB2Seqs, 64, *171*, *177*
DECIPHER *(*DECIPHER-package*)*, 4
DECIPHER-package, 4
deltaGrules, *40*, 66
deltaGrulesRNA, 67, *159*
deltaHrules, 68
deltaHrulesRNA, 69
deltaSrules, 70
deltaSrulesRNA, 71
DesignArray, *32*, 72, *152*
DesignPrimers, *31*, *44*, 74, *84*, *186*
DesignProbes, *42*, 78
DesignSignatures, *31*, *44*, *77*, 81, *90*, *144*
DetectRepeats, 85
DigestDNA, *84*, 89
Disambiguate, *41*, *44*, *52*, *84*, 90
DistanceMatrix, *46–48*, 91, *150*, *192*, *193*

ExtractGenes, 98, *103*, *105*, *111*, *201*

FindChimeras, *60*, 99
FindGenes, *98*, 102, *111*, *201*
FindNonCoding, *103*, 104, *133*, *153*
FindSynteny, *7*, *26*, *86*, *87*, 105, *181*

FormGroups, 108, *115*

Genes, 110
Genes-class *(*Genes*)*, 110

HEC_MI, 112
HEC_MI1, *164*
HEC_MI1 *(*HEC_MI*)*, 112
HEC_MI2, *164*
HEC_MI2 *(*HEC_MI*)*, 112

IdConsensus, *52*, 113
IdentifyByRank, *109*, 114
IdLengths, 116, *185*
IdTaxa, 117, *136, 137*, *183*
IndexSeqs, *17*, 120, *132*, *174*
InferDemography, 123, *128*, *131*
InferRecombination, 125, *125*, *131*
InferSelection, *125*, 128, *128*
InvertedIndex, 131
InvertedIndex-class *(*InvertedIndex*)*, 131

LearnNonCoding, *105*, 132, *153*
LearnTaxa, *7*, *119*, 134, *183*, *187*

MapCharacters, 138, *193*
MaskAlignment, 140
MeltDNA, *31*, *84*, 143
MIQS, *21*, 145
MMLSUM, 146
MODELS, *95*, 147, *193*

NNLS, *32*, *74*, 151
NonCoding, 152
NonCoding-class *(*NonCoding*)*, 152
NonCodingRNA, 153
NonCodingRNA_Archaea *(*NonCodingRNA*)*, 153
NonCodingRNA_Bacteria *(*NonCodingRNA*)*, 153
NonCodingRNA_Eukarya *(*NonCodingRNA*)*, 153

OrientNucleotides, *58*, 154

pairs.Synteny *(*Synteny*)*, 179
PAM, 156
PFASUM, *7*, *11*, *13*, *21*, *58*, 157, *169*
plot.Genes *(*Genes*)*, 110
plot.Synteny *(*Synteny*)*, 179
plot.Taxa *(*Taxa*)*, 182
PredictDBN, 158, *164*
PredictHEC, *161*, 162
print.Genes *(*Genes*)*, 110
print.InvertedIndex *(*InvertedIndex*)*, 131
print.NonCoding *(*NonCoding*)*, 152
print.Synteny *(*Synteny*)*, 179
print.Taxa *(*Taxa*)*, 182

ReadDendrogram, *25*, *62*, 164, *193*, *200*, *204*
RemoveGaps, 165
RESTRICTION_ENZYMES, *82*, *84*, *90*, 167

ScoreAlignment, *87*, 167
SearchDB, *8*, 170, *177*
SearchIndex, *17*, *122*, *132*, 172
Seqs2DB, *8*, *60*, *114*, *171*, 175
StaggerAlignment, *11*, *25*, 177
Synteny, 179
Synteny-class *(*Synteny*)*, 179

Taxa, 182
Taxa-class *(*Taxa*)*, 182
TerminalChar, 184
TileSeqs, *42*, *77*, *80*, 185
TrainingSet_16S, 187
TreeLine *(*Treeline*)*, 188
Treeline, *25*, *48*, *54*, *62*, *95*, *139*, *142*, *150*, *165*, *177, 178*, 188, *200*, *203, 204*
TrimDNA, 196

WriteDendrogram, *62*, *165*, *193*, 199, *204*
WriteGenes, *98*, *103*, *105*, *111*, 201

Zipline, *62*, *193*, 202