Package 'DECIPHER'

September 24, 2012

Type Package

Title Database Enabled Code for Ideal Probe Hybridization Employing R

Version 1.2.0

Date 2012-03-03

Author Erik Wright

Maintainer Erik Wright <DECIPHER@cae.wisc.edu>

biocViews Clustering, Genetics, Sequencing, Infrastructure,DataImport, Visualization, Microarray, QualityControl

Description A toolset that assist in the design of hybridization probes.

Depends R (>= 2.13.0), Biostrings (>= 2.16), RSQLite (>= 0.9),IRanges, stats

Imports Biostrings, RSQLite, IRanges, stats

LinkingTo Biostrings, RSQLite, IRanges, stats

License GPL-3

LazyData yes

R topics documented:

DECIPHER-package *Database Enabled Code for Ideal Probe Hybridization Employing R*

Description

Database Enabled Code for Ideal Probe Hybridization Employing R (DECIPHER) is a software toolset that can be used for deciphering and managing DNA sequences efficiently using the R statistical programming language. The program is designed to be used with non-destructive workflows that guide the user through the process of importing, maintaining, analyzing, manipulating, and exporting a massive amount of DNA sequences. Some functionality of the program is provided online through web tools. DECIPHER is an ongoing project in the Environmental Engineering Department at the University of Wisconsin Madison and is freely available for download.

Details

Package:	DECIPHER
Type:	Package
Version:	1.3.5
Date:	2011-07-22
Depends:	R (>= 2.13.0), Biostrings (>= 2.16), RSQLite (>= 0.9), IRanges, stats
Imports:	Biostrings, RSQLite, IRanges, stats
LinkingTo:	Biostrings, RSQLite, IRanges, stats
License:	GPL-3
LazyLoad:	yes

Index:

Add2DB	Add Data To A Database
BrowseDB	View A Database Table In A Web Browser
BrowseSequences	View Sequences In A Web Browser
ConsensusSequence	Create A Consensus Sequence
CreateChimeras	Creates Artificial Chimeras
DB2FASTA	Export Database to FASTA File
DistanceMatrix	Calculate the Distance Between DNA Sequences
FindChimeras	Find Chimeras In A Sequence Database
FormGroups	Forms Groups By Rank
IdClusters	Cluster Sequences By Distance
IdConsensus	Create Consensus Sequences by Groups
IdLengths	Determine the Number of Bases and Nonbases In Each Sequence
IdentifyByRank	Update Identifier To Level of Taxonomic Rank
SearchDB	Obtain Specific Sequences from A Database

```
Seqs2DB                 Add Sequences from Text File to Database
TerminalChar            Determine the Number of Terminal Gaps
```

Author(s)

Erik Wright

Maintainer: Erik Wright <DECIPHER@cae.wisc.edu>

`Add2DB` *Add Data To A Database*

Description

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

Usage

```
Add2DB(myData,
       dbFile,
       tblName = "DNA",
       verbose = TRUE,
       ...)
```

Arguments

`myData`	Data frame containing information to be added to the `dbFile`.
`dbFile`	A SQLite connection object or a character string specifying the path to the database file.
`tblName`	Character string specifying the table in which to add the data.
`verbose`	Logical indicating whether to display each query as it is sent to the database.
`...`	Additional expressions to add as part of a where clause in the query. Further arguments provided in ... will be added to the query separated by `" and "` as part of the where clause.

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.

Author(s)

Erik Wright `<DECIPHER@cae.wisc.edu>`

See Also

`Seqs2DB`, `SearchDB`, `BrowseDB`

Examples

```
# Create a sequence database
gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
dbConn <- dbConnect(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)
dbDisconnect(dbConn)
```

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=file.path(tempdir(), "db.html"),
         tblName = "DNA",
         identifier = "",
         limit = -1,
         orderBy = "row_names",
         maxChars = 50,
         ...)
```

Arguments

dbFile	A SQLite connection object or a character string specifying the path to the database file.
htmlFile	Character string giving the location where the html file should be written.
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.
...	Additional expressions to add as part of a where clause in the query. Further arguments provided in ... will be added to the query separated by " and " as part of the where clause.

Value

Creates a table containing all the fields of the database table and opens it in the web browser for easy viewing.

Returns TRUE if the html file was written successfully.

Note

If viewing a table containing sequences, the sequences are purposefully not shown in the output.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

BrowseSequences

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
BrowseDB(db)
```

BrowseSequences *View Sequences In A Web Browser*

Description

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

Usage

```
BrowseSequences(myDNAStringSet,
                htmlFile = paste(tempdir(), "/dna.html", sep = ""),
                colorBases=FALSE,
                ...)
```

Arguments

myDNAStringSet	A DNAStringSet object of sequences.
htmlFile	Character string giving the location where the html file should be written.
colorBases	Logical specifying whether to color each type of base (A, C, G, and T) the same color.
...	Additional arguments to be passed directly to ConsensusSequence.

Details

Some web browsers cannot quickly display a large amount data, so it is recommended to use color = FALSE (the default) when viewing a large DNAStringSet.

Value

Creates an html file containing sequence data and opens it in a web browser for easy viewing. The viewer has the sequence name on the left, position legend on the top, number of characters on the right, and consensus sequence on the bottom.

Returns TRUE if the html file was written successfully.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

BrowseDB

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
dna <- SearchDB(db)
BrowseSequences(dna[1:5], colorBases=TRUE)
```

CalculateEfficiencyArray

Calculates the Efficiency of Probe/Target Sequence Pairs

Description

Calculates the Gibb's 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:

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

Probe and target input sequences must be entered in pairwise alignment, such as that given by `pairwiseAlignment`. Only "A", "C", "G", "T", and "-" characters are permitted in the probe sequence.

If deltaGrules is NULL then the rules defined in `data(deltaGrules)` are used.

Value

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

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

References

Coming soon!

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")
```

ConsensusSequence *Create A Consensus Sequence*

Description

Forms a consensus sequence representing a set of sequences.

Usage

```
ConsensusSequence(myDNAStringSet,
                  threshold = 0.05,
                  ambiguity = TRUE,
                  noConsensusChar = "N",
                  minInformation = 0.75,
                  ignoreNonBases = FALSE,
                  includeTerminalGaps = FALSE,
                  verbose = TRUE)
```

Arguments

myDNAStringSet	A DNAStringSet object of aligned sequences.
threshold	Maximum fraction of sequence information that may be lost in forming the consensus.
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 DNA_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.
ignoreNonBases	Logical specifying whether to count gap ("-") or mask ("+") characters towards the consensus.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" characters on each end of the sequence) into the formation of consensus.
verbose	Logical indicating whether to print the elapsed time upon completion.

Details

Two key parameters control the degree of consensus. The default threshold (0.05) indicates that at least 95% of sequence information will be represented by the consensus sequence. The default minInformation (0.75) specifies that at least 75% of sequences must contain the information in the consensus, otherwise the noConsensusChar is used.

If ambiguity = TRUE (the default) then degeneracy codes are split between their respective bases according to the IUPAC_CODE_MAP. For example, an "R" would count as half an "A" and half a "G". If ambiguity = FALSE then degeneracy codes are not considered in forming the consensus. If includeNonBases = TRUE (the default) then gap ("-") and mask ("+") characters are counted towards the consensus, otherwise they are omitted from development of the consensus.

Value

A DNAStringSet with a single consensus sequence.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

IdConsensus, Seqs2DB

Examples

```
dna <- DNAStringSet(c("ANGCT-","-ACCT-"))
ConsensusSequence(dna)
# returns "ANSCT-"
```

CreateChimeras *Creates 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,
               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.
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 the chimeras are included at the end of the result. The parents are made to be 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 <DECIPHER@cae.wisc.edu>

See Also

FindChimeras, Seqs2DB

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
dna <- SearchDB(db)
chims <- CreateChimeras(dna)
BrowseSequences(chims)
```

DB2FASTA *Export Database Sequences to FASTA File*

Description

Exports a database containing sequences to a FASTA formatted file of sequences.

Usage

```
DB2FASTA(file,
         dbFile,
         tblName = "DNA",
         identifier = "",
         limit = -1,
         replaceChar = NULL,
         orderBy = "row_names",
         append = FALSE,
         comments = TRUE,
         removeGaps = "none",
         verbose = TRUE,
         ...)
```

Arguments

file	Character string giving the location where the FASTA file should be written.
dbFile	A SQLite connection object or a character string specifying the path to the 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.
limit	Number of results to display. The default (-1) does not limit the number of results.
replaceChar	Optional character used to replace any sequence characters not present in the DNA_ALPHABET. If NULL (the default) then no replacement occurs and the sequences are exported identical to how they were upon import.
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.
append	Logical indicating whether to append the results to the existing file.
comments	Logical specifying whether to add the value of any database fields into the FASTA record description separated by semicolons.
removeGaps	Determines how gaps are removed in the sequences. This should be (an unambiguous abbreviation of) one of "none", "all" or "common".
verbose	Logical indicating whether to display status.
...	Additional expressions to add as part of a where clause in the query. Further arguments provided in ... will be added to the query separated by " and " as part of the where clause.

Value

Writes a FASTA formatted file containing the sequences in the database.

Returns TRUE if the file was written successfully.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
tf <- tempfile()
DB2FASTA(tf, db, l=10)
file.show(tf)
unlink(tf)
```

`deltaGrules` *An 8-dimensional array containing the free energy of hybridization of probe/target quadruplets.*

Description

The 8D array works with four adjacent base pairs of the probe and target sequence at a time. Each dimension has five elements defining the residue at that position ("A", "C", "G", "T", or "-"). The array contains the standard Gibb's 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 4 probe positions from 5' to 3'. The fifth to eighth dimensions correspond to the 4 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

Coming soon!

Examples

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

DistanceMatrix *Calculate the Distance Between DNA Sequences*

Description

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

Usage

```
DistanceMatrix(myDNAStringSet,
               includeTerminalGaps = FALSE,
               penalizeGapLetterMatches = TRUE,
               penalizeGapGapMatches = FALSE,
               removeDuplicates = FALSE,
               correction = "none",
               verbose = TRUE)
```

Arguments

myDNAStringSet	A DNAStringSet object of aligned sequences.
includeTerminalGaps	Logical specifying whether or not to include terminal gaps ("-" characters on each end of the sequence) into the calculation of distance.
penalizeGapLetterMatches	Logical specifying whether or not to consider gap-to-letter matches as mismatches.
penalizeGapGapMatches	Logical specifying whether or not to consider gap-to-gap matches as mismatches.
removeDuplicates	Logical specifying whether to remove any identical sequences from the DNAStringSet before calculating distance. If FALSE (the default) then the distance matrix is calculated with the entire DNAStringSet provided as input.
correction	The substitution model used for distance correction. This should be (an unambiguous abbreviation of) one of "none" or "Jukes-Cantor".
verbose	Logical indicating whether to display progress.

Details

The uncorrected distance matrix represents the percent distance between each of the sequences in the DNAStringSet. Ambiguity can be represented using the characters of the IUPAC_CODE_MAP. For example, the distance between an 'N' and any other base is zero.

If includeTerminalGaps = FALSE then terminal gaps are not included in sequence length. This can be faster since only the positions common to each two sequences are compared. If removeDuplicates = TRUE then the distance matrix will only represent unique sequences in the DNAStringSet. This is can be faster because less sequences need to be compared. For example, if only two sequences in the set are exact duplicates then one is removed and the distance is calculated on the remaining set. Note that the distance matrix can still contain values of 100% after removing duplicates because only exact duplicates are removed without taking into account ambiguous matches represented by the IUPAC_CODE_MAP or the treatment of gaps.

The elements of the distance matrix can be referenced by dimnames corresponding to the names of the DNAStringSet. Additionally, an attribute named "correction" specifying the method of correction used can be accessed using the function attr.

Value

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 DNAStringSet. Sequences with no overlapping positions in the alignment are given a value of NA.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

IdClusters

Examples

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

# compare union of internal ranges:
dna <- DNAStringSet(c("ANGCT-","-ACCT-"))
d <- DistanceMatrix(dna, includeTerminalGaps=TRUE)
# d[1,2] is now 2 bases in 5 = 0.40

# compare the entire sequence ranges:
dna <- DNAStringSet(c("ANGCT-","-ACCT-"))
d <- DistanceMatrix(dna, includeTerminalGaps=TRUE,
                    penalizeGapGapMatches=TRUE)
# d[1,2] is now 3 bases in 6 = 0.50
```

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 = "DNA",
             dbFileReference,
             batchSize = 100,
             minNumFragments = 20000,
             tb.width = 5,
```

```
              multiplier = 20,
              minLength = 70,
              minCoverage = 0.6,
              overlap = 200,
              minSuspectFragments = 6,
              showPercentCoverage = FALSE,
              add2tbl = FALSE,
              maxGroupSize = -1,
              verbose = TRUE)
```

Arguments

dbFile	A SQLite connection object or a character string specifying the path to the database file to be checked for chimeric sequences.
tblName	Character string specifying the table in which to check for chimeras.
dbFileReference	A SQLite connection object or a character string specifying the path to the reference database file of (presumed to be) good quality sequences. A 16S reference database is available from DECIPHER.cee.wisc.edu.
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.
verbose	Logical indicating whether to display progress.

Details

The algorithm works by finding suspect 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 optimal parameters that are different than the defaults. These are: `minLength = 40`, 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 `dbFile`.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

References

ES Wright et al. (2011) "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`

Examples

```
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 DECIPHER.cee.wisc.edu
chimeras <- FindChimeras(db, dbFileReference=db)
```

FormGroups *Forms Groups By Rank*

Description

Agglomerates sequences into groups in a certain size range based on taxonomic rank.

Usage

```
FormGroups(dbFile,
           tblName = "DNA",
           goalSize = 1000,
           minGroupSize = 500,
           maxGroupSize = 10000,
           add2tbl = FALSE,
           verbose = TRUE)
```

Arguments

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

Form groups uses the rank field in the dbFile table to group sequences with similar taxonomic rank. Requires that rank information be present in the tblName, such as that created when importing sequences from a GenBank file.

Beginning with the least common ranks, the algorithm agglomerates groups with similar ranks 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.

Value

Returns a data.frame of rank and id for each group. If add2tbl is not FALSE then the tblName is updated with the group as the identifier.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

IdentifyByRank

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
g <- FormGroups(db, goalSize=10, minGroupSize=5, maxGroupSize=20)
```

IdClusters *Cluster Sequences By Distance*

Description

Groups the sequences represented by a distance matrix into clusters of similarity.

Usage

```
IdClusters(myDistMatrix,
           method = "UPGMA",
           cutoff = -Inf,
           showPlot = FALSE,
           asDendrogram = FALSE,
           myDNAStringSet = NULL,
           add2tbl = FALSE,
           dbFile = NULL,
           verbose = TRUE)
```

Arguments

myDistMatrix	A symmetric N x N distance matrix with the values of dissimilarity between N sequences.
method	An agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "complete", "single", "UPGMA", "average", "NJ" or "ML".
cutoff	A vector with the maximum branch length separating the sequences in the same cluster. If asDendrogram=TRUE then only one cutoff may be specified.
showPlot	Logical specifying whether or not to plot the resulting dendrogram.
asDendrogram	Logical. If TRUE the object returned is of class dendrogram.
myDNAStringSet	DNAStringSet used in the creation of myDistMatrix. Only necessary if method="ML".
add2tbl	Logical or a character string specifying the table name in which to add the result.
dbFile	A connection to a SQLite database or character string giving the path to the database file. Only necessary if add2tbl is not FALSE.
verbose	Logical indicating whether to display progress.

Details

Groups the input sequences into clusters using a set dissimilarities representing the distance between N sequences. Initially a phylogenetic tree is formed using the specified method. Then each leaf (sequence) of the tree is assigned to a cluster based on its branch lengths to the other leaves (sequences).

A number of different clustering methods are provided. The method (complete assigns clusters using complete-linkage so that sequences in the same cluster are no more than cutoff percent apart. The method single assigns clusters using single-linkage so that sequences in the same cluster are within cutoff of at least one other sequence in the same cluster. UPGMA or average (the default) assigns 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 relatives that do not appear to be closely related.

NJ uses the Neighbor-Joining method proposed by Saitou and Nei that does not assume lineages evolve at the same rate (the molecular clock hypothesis). The NJ method is typically the most phylogenetically accurate of the above distance based methods. ML creates a neighbor-joining tree and then prints the negative log likelihood of the tree. Presently ML does not adjust the neighbor joining tree to maximize its likelihood.

If a add2tbl=TRUE then the resulting data.frame is added/updated into column(s) of the default table "DNA" in dbFile. If add2tbl is a character string then the result is added to the specified table name in dbFile. The added/updated column names are printed if verbose=TRUE.

Value

If asDendrogram=FALSE (the default), returns a data.frame with a column for each cutoff specified. The row.names of the data.frame correspond to the dimnames of myDistMatrix. Each one of N sequences is assigned to one of M clusters. If asDendrogram=TRUE, returns an object of class dendrogram that can be used for further manipulation and plotting. Leaves of the dendrogram are randomly colored by cluster number.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

References

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17(6)**, 368-376

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.

See Also

DistanceMatrix, Add2DB

Examples

```
# using the matrix from the original paper by Saitou and Nei
m <- matrix(0,8,8)
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] <- c(8)

# returns an object of class "dendrogram"
myClusters <- IdClusters(m, cutoff=10, method="NJ", showPlot=TRUE, asDendrogram=TRUE)

# example of specifying a cutoff
# returns a data frame
IdClusters(m, cutoff=c(2,6,10,20))
```

IdConsensus *Create Consensus Sequences by Groups*

Description

Forms a consensus sequence representing the sequences in each group.

Usage

```
IdConsensus(dbFile,
            tblName = "DNA",
            identifier = "",
            colName = "cluster",
            add2tbl = FALSE,
            verbose = TRUE,
            ...)
```

Arguments

dbFile	A SQLite connection object or a character string specifying the path to the 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.
colName	Column containing the group name of each sequence.
add2tbl	Logical or a character string specifying the table name in which to add the result.
verbose	Logical indicating whether to display progress.
...	Additional arguments to be passed directly to ConsensusSequence.

Details

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

Value

A DNAStringSet object containing the consensus sequence for each group. The names of the DNAStringSet contain the number of sequences and name of each group.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

Seqs2DB

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
con <- IdConsensus(db, colName="id")
BrowseSequences(con, colorBases=TRUE)
```

`IdentifyByRank` *Identify By Taxonomic Rank*

Description

Identifies sequences by a specific level of their taxonomic rank.

Usage

```
IdentifyByRank(dbFile,
               tblName = "DNA",
               level = 3,
               add2tbl = FALSE,
               verbose = TRUE)
```

Arguments

`dbFile`	A SQLite connection object or a character string specifying the path to the database file.
`tblName`	Character string specifying the table where the rank information is located.
`level`	Level of the taxonomic rank.
`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

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

If the specified level of rank does not exist then the closest rank is chosen. This makes it possible to determine the lowest level classification (e.g., genus) by specifying `level = 100`.

Value

A `data.frame` with the `rank` and corresponding identifier as `"id"`.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

`FormGroups`

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
ids <- IdentifyByRank(db)
```

IdLengths *Determine the Number of Bases, Nonbases, and Width of Each Sequence*

Description

Counts the number of bases (A, C, G, T) and ambiguities/degeneracies in each sequence.

Usage

```
IdLengths(dbFile,
          tblName = "DNA",
          identifier = "",
          add2tbl = FALSE,
          verbose = TRUE)
```

Arguments

dbFile	A SQLite connection object or a character string specifying the path to the 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 "" then all identifiers are selected.
add2tbl	Logical or a character string specifying the table name in which to add the result.
verbose	Logical indicating whether to display progress.

Value

A `data.frame` with the number of `bases`, `nonbases`, and `width` of each sequence. The `width` is defined as the sum of `bases` and `nonbases` in each sequence. The `row.names` of the `data.frame` correspond to the "row_names" in the `tblName` of the `dbFile`.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

Add2DB

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
l <- IdLengths(db)
```

SearchDB *Obtain Specific Sequences from A Database*

Description

Returns the set of sequences meeting the search criteria.

Usage

```
SearchDB(dbFile,
         tblName = "DNA",
         identifier = "",
         limit = -1,
         replaceChar = "-",
         orderBy = "row_names",
         countOnly = FALSE,
         removeGaps = "none",
         verbose = TRUE,
         ...)
```

Arguments

dbFile	A SQLite connection object or a character string specifying the path to the 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 "" then all identifiers are selected.
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 DNA_ALPHABET.
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 are removed in the sequences. This should be (an unambiguous abbreviation of) one of "none", "all" or "common".
verbose	Logical indicating whether to display queries as they are sent to the database.
...	Additional expressions to add as part of a where clause in the query. Further arguments provided in ... will be added to the query separated by " and " as part of the where clause.

Details

If RNA is present in the database then all U's are converted to T's before creating the DNAStringSet.

Value

A DNAStringSet with the sequences that meet the specified criteria. The names of the DNAStringSet correspond to the value in the "row_names" column of the database.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

Seqs2DB, DB2FASTA

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
dna <- SearchDB(db)
```

Seqs2DB *Add Sequences from Text File to Database*

Description

Adds sequences to a database.

Usage

```
Seqs2DB(seqs,
        type,
        dbFile,
        identifier,
        tblName = "DNA",
        chunkSize = 99999,
        replaceTbl = FALSE,
        verbose = TRUE)
```

Arguments

seqs	Either a character string specifying the file path to the file containing the sequences, or a DNAStringSet object.
type	The type of sequences being imported. This should be (an unambiguous abbreviation of) one of "FASTA", "GenBank", or "DNAStringSet".
dbFile	A SQLite connection object or a character string specifying the path to the database file. If the dbFile does not exist then a new database is created at this location.
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 lines of the seqs to read at a time. For very large sequence files, using 1e7 results in a quicker import than the default (99999), but only if enough memory is available.
replaceTbl	Logical. If FALSE (the default) then the sequences are appended to any already existing in the table. If TRUE then any sequences already in the table are overwritten.
verbose	Logical indicating whether to display each query as it is sent to the database.

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.

Value

The total number of sequences in the database table is returned after import.

Author(s)

Erik Wright <DECIPHER@cae.wisc.edu>

See Also

SearchDB, DB2FASTA

Examples

```
gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
dbConn <- dbConnect(SQLite(), ":memory:")
Seqs2DB(gen, "GenBank", dbConn, "Bacteria")
BrowseDB(dbConn)
dbDisconnect(dbConn)
```

TerminalChar *Determine the Number of Terminal Characters*

Description

Counts the number of terminal characters for every sequence in a `DNAStringSet`. Terminal characters are defined as a specific character repeated at the beginning and end of a sequence.

Usage

```
TerminalChar(myDNAStringSet,
             char = "-")
```

Arguments

`myDNAStringSet`	A `DNAStringSet` object of sequences.
`char`	A single character giving the terminal character to count.

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 <DECIPHER@cae.wisc.edu>

See Also

IdLengths

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package="DECIPHER")
dna <- SearchDB(db)
t <- TerminalChar(dna)
```

Index