Package 'SynExtend'

October 11, 2022

Type Package

Title Tools for Working With Synteny Objects

Version 1.8.0

biocViews Genetics, Clustering, ComparativeGenomics, DataImport

Description Shared order between genomic sequences provide a great deal of information. Synteny objects produced by the R package DECIPHER provides quantitative information about that shared order. SynExtend provides tools for extracting information from Synteny objects.

Depends R (>= 4.1.0), DECIPHER (>= 2.20.0)

Imports methods, Biostrings, S4Vectors, IRanges, utils, stats, parallel, graphics, grDevices

Suggests BiocStyle, knitr, rtracklayer, igraph, markdown, rmarkdown

License GPL-3

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/SynExtend

git_branch RELEASE_3_15

git_last_commit ada2504

git_last_commit_date 2022-04-26

Date/Publication 2022-10-11

Author Nicholas Cooley [aut, cre] (<https://orcid.org/0000-0002-6029-304X>), Aidan Lakshman [aut, ctb] (<https://orcid.org/0000-0002-9465-6785>), Adelle Fernando [ctb], Erik Wright [aut]

Maintainer Nicholas Cooley <npc19@pitt.edu>

R topics documented:

`BlastSeqs` *Run BLAST queries from R*

Description

Wrapper to run BLAST queries using the commandline BLAST tool directly from R. Can operate on an `XStringSet` or a `FASTA` file.

This function requires the BLAST+ commandline tools, which can be downloaded here.

Usage

```
BlastSeqs(seqs, BlastDB,
              blastType=c('blastn', 'blastp', 'tblastn', 'blastx', 'tblastx'),
              extraArgs='', verbose=TRUE)
```

Arguments

`seqs`	Sequence(s) to run BLAST query on. This can be either an `XStringSet` or a path to a FASTA file.
`BlastDB`	Path to FASTA file in a pre-built BLAST Database. These can be built using the commandline `makeblastdb` function from BLAST+. For more information on building BLAST DBs, see here.
`blastType`	Type of BLAST query to run. See 'Details' for more information on available types.
`extraArgs`	Additional arguments to be passed to the BLAST query excuted on the command line. This should be a single character string.
`verbose`	Should output be displayed?

Details

BLAST implements multiple types of search. Available types are the following:

- `blastn`: Nucleotide sequences against database of nucleotide sequences
- `blastp`: Protein sequences against database of protein sequences
- `tblastn`: Translates nucleotide sequences to protein sequences, then queries against database of protein sequences
- `blastx`: Queries protein sequences against database of nucleotides translated into protein sequences
- `tblastx`: Translates nucleotide sequences to protein sequences, then queries against database of nucleotides translated into protein sequences

Different BLAST queries require different inputs. The function will throw an error if the input data does not match expected input for the requested query type.

Input data for `blastn`, `tblastn`, and `tblastx` should be nucleotide data.

Input data for `blastp` and `blastx` should be amino acid data.

Value

Returns a data frame (data.frame) of results of the BLAST query.

Note

Future release will add ability to create a BLAST database from input data directly in R.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

Examples

```
#
```

BlockExpansion *Attempt to expand blocks of paired features in a* PairSummaries *object.*

Description

Attempt to expand blocks of paired features in a PairSummaries object.

Usage

```
BlockExpansion(Pairs,
               GapTolerance = 4L,
               DropSingletons = FALSE,
               Criteria = "PID",
               Floor = 0.5,
               NewPairsOnly = TRUE,
               DBPATH,
               Verbose = FALSE)
```

Arguments

Pairs	An object of class PairSummaries.
GapTolerance	Integer value indicating the diff between feature IDs that can be tolerated to view features as part of the same block. Set by default to 4L, implying that a single feature missing in a run of pairs will not cause the block to be split. Setting to 3L would imply that a diff of 3 between features, or a gap of 2 features, can be viewed as those features being part of the same block.
DropSingletons	Ignore solo pairs when planning expansion routes. Set to FALSE by default.
Criteria	Either "PID" or "Score", indicating which metric to use to keep or reject pairs.
Floor	Lower PID limit for keeping a pair that was evaluated during expansion.
NewPairsOnly	Logical indicating whether or not to return only the pairs that were kept from all expansion attempts, or to return a PairSummaries object with the new pairs folded in.
DBPATH	A file or connection pointing to the DECIPHER database supplied to FindSynteny for the original map construction.
Verbose	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

BlockExpansion uses a naive expansion algorithm to attempt to fill in gaps in blocks of paired features and to attempt to expand blocks of paired features.

Value

An object of class PairSummaries.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

PairSummaries, NucleotideOverlap, link{SubSetPairs}, FindSynteny

Examples

```
DBPATH <- system.file("extdata",
                      "Endosymbionts.sqlite",
                      package = "SynExtend")

data("Endosymbionts_Pairs01", package = "SynExtend")
Pairs02 <- BlockExpansion(Pairs = Endosymbionts_Pairs01,
                          NewPairsOnly = FALSE,
                          DBPATH = DBPATH,
                          Verbose = TRUE)
```

BlockReconciliation *Rejection scheme for asyntenic predicted pairs*

Description

Take in a PairSummaries object and reject predicted pairs that conflict with syntenic blocks either locally or globally.

Usage

```
BlockReconciliation(Pairs,
                    ConservativeRejection = TRUE,
                    Precedent = "Size",
                    PIDThreshold = NULL,
                    SCOREThreshold = NULL,
                    Verbose = FALSE)
```

Arguments

Pairs	A PairSummaries object.
ConservativeRejection	A logical defaulting to TRUE. By default only pairs that conflict within a syntenic block will be rejected. When FALSE any conflict will cause the rejection of the pair in the smaller block.
Precedent	A character vector of length 1, defaulting to "Size". Selector for whether function attempts to reconcile with block size as precedent, or mean block PID as precedent. Currently "Metric" will select mean block PID to set block precedent. Blocks of size 1 cannot reject other blocks. The default behavior causes

the rejection of any set of predicted pairs that conflict with a larger block of predicted pairs. Switching to "Metric" changes this behavior to any block of size 2 or greater will reject any predicted pair that both conflicts with the current block, and is part of a block with a lower mean PID.

`PIDThreshold` Defaults to `NULL`, a numeric of length 1 can be used to retain pairs that would otherwise be rejected. Pairs that would otherwise be rejected that have a `PID` >= `PIDThreshold` will be retained.

`SCOREThreshold` Defaults to `NULL`, a numeric of length 1 can be used retain pairs that would otherwise be rejected. Pairs that would otherwise be rejected that have a `SCORE` >= `SCOREThreshold` will be retained.

`Verbose` Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

If a given `PairSummaries` object contains predicted pairs that conflict, i.e. imply paralogy, or an "incorrect" and a "correct" ortholog prediction, these predictions will be reconciled. The function scrolls through pairs based on the size of the syntenic block that they are part of, from largest to smallest. When `ConservativeRejection` is `TRUE` only predicted pairs that exist within the syntenic block "space" will be removed, this option leaves room for conflicting predictions to remain if they are non-local to each other, or are on different indices. When `ConservativeRejection` is `FALSE` any pair that conflicts with a larger syntenic block will be rejected. This option forces only 1-1 feature pairings, for features are part of any syntenic block. Predicted pairs that represent a syntenic block size of 1 feature will not reject other pairs. `PIDThreshold` and `SCOREThreshold` can be used to retain pairs that would otherwise be rejected based on available assessments of their pairwise alignment.

Value

A data.frame of class "data.frame" and "PairSummaries" of paired genes that are connected by syntenic hits. Contains columns describing the k-mers that link the pair. Columns "p1" and "p2" give the location ids of the the genes in the pair in the form "DatabaseIdentifier_ContigIdentifier_GeneIdentifier". "ExactMatch" provides an integer representing the exact number of nucleotides contained in the linking k-mers. "TotalKmers" provides an integer describing the number of distinct k-mers linking the pair. "MaxKmer" provides an integer describing the largest k-mer that links the pair. A column titled "Consensus" provides a value between zero and 1 indicating whether the kmers that link a pair of features are in the same position in each feature, with 1 indicating they are in exactly the same position and 0 indicating they are in as different a position as is possible. The "Adjacent" column provides an integer value ranging between 0 and 2 denoting whether a feature pair's direct neighbors are also paired. Gap filled pairs neither have neighbors, or are included as neighbors. The "TetDist" column provides the euclidean distance between oligonucleotide - of size 4 - frequences between predicted pairs. "PIDType" provides a character vector with values of "NT" where either of the pair indicates it is not a translatable sequence or "AA" where both sequences are translatable. If users choose to perform pairwise alignments there will be a "PID" column providing a numeric describing the percent identity between the two sequences. If users choose to predict PIDs using their own, or a provided model, a "PredictedPID" column will be provided.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class, PairSummaries

Examples

```
data("Endosymbionts_Pairs02", package = "SynExtend")
Pairs03 <- BlockReconciliation(Pairs = Endosymbionts_Pairs02,
                               ConservativeRejection = FALSE,
                               Verbose = TRUE)
```

BuiltInEnsembles *Pretrained ProtWeaver Ensemble Models*

Description

ProtWeaver has best performance with an ensemble method combining individual evidence streams. This data file provides pretrained models for ease of use. These models are trained on genes from *Streptomyces* species.

These models are used internally if the user does not provide their own model, and aren't explicitly designed to be accessed by the user.

See the examples for how to train your own ensemble model.

Usage

```
data("BuiltInEnsembles")
```

Format

The data contain a list of objects of class glm.

Examples

```
## Training own ensemble method to avoid
## using built-ins

exData <- get(data("ExampleStreptomycesData"))
pw <- ProtWeaver(exData$Genes[1:50])
datavals <- predict(pw, NoPrediction=TRUE)

# Make sure the actual values correspond to the right pairs!
# This example just picks random numbers
# Do not do this for your own models
actual_values <- sample(c(0,1), nrow(datavals), replace=TRUE)
datavals[,'y'] <- actual_values
```

```
myModel <- glm(y~., datavals[,-c(1,2)], family='binomial')

predictionPW <- ProtWeaver(exData$Genes[51:60])
predict(predictionPW,
          PretrainedModel=myModel)
```

DisjointSet *Return single linkage clusters from* PairSummaries *objects.*

Description

Takes in a PairSummaries object and return a list of identifiers organized into single linkage clusters.

Usage

```
DisjointSet(Pairs,
            Verbose = FALSE)
```

Arguments

Pairs	A PairSummaries object.
Verbose	Logical indicating whether to print progress bars and messages. Defaults to FALSE.

Details

Takes in a PairSummaries object and return a list of identifiers organized into single linkage clusters.

Value

Returns a list of character vectors representing IDs of sequence features, typically genes.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class, PairSummaries, FindSets

Examples

```
data("Endosymbionts_Pairs03", package = "SynExtend")

Sets <- DisjointSet(Pairs = Endosymbionts_Pairs03,
                    Verbose = TRUE)
```

Endosymbionts_GeneCalls

Example genecalls

Description

A named list of DataFrames.

Usage

```
data("Endosymbionts_GeneCalls")
```

Format

A named list.

Details

Example genecalls.

Examples

```
data(Endosymbionts_GeneCalls)
```

Endosymbionts_LinkedFeatures

Example synteny links

Description

An object of class LinkedPairs.

Usage

```
data("Endosymbionts_LinkedFeatures")
```

Format

An object of class LinkedPairs.

Details

An object of class LinkedPairs.

Examples

```
data(Endosymbionts_LinkedFeatures)
```

Endosymbionts_Pairs01 *Example predicted pairs*

Description

An object of class `PairSummaries`.

Usage

```
data("Endosymbionts_Pairs01")
```

Format

An object of class `PairSummaries`.

Details

An object of class `PairSummaries`.

Examples

```
data(Endosymbionts_Pairs01)
```

Endosymbionts_Pairs02 *Example predicted pairs*

Description

An object of class `PairSummaries` where blocks have been expanded.

Usage

```
data("Endosymbionts_Pairs02")
```

Format

An object of class `PairSummaries`.

Details

An object of class `PairSummaries`.

Examples

```
data(Endosymbionts_Pairs02)
```

Endosymbionts_Pairs03 *Example predicted pairs*

Description

An object of class `PairSummaries` where blocks have been expanded and competitors have been rejected.

Usage

```
data("Endosymbionts_Pairs03")
```

Format

An object of class `PairSummaries`.

Details

An object of class `PairSummaries`.

Examples

```
data(Endosymbionts_Pairs03)
```

Endosymbionts_Sets *A list of disjoint sets.*

Description

A named list of disjoint sets representing hypothetical COGs.

Usage

```
data("Endosymbionts_Sets")
```

Format

A named list of disjoint sets representing hypothetical COGs.

Details

A named list of disjoint sets representing hypothetical COGs.

Examples

```
data(Endosymbionts_Sets)
```

Endosymbionts_Synteny *A synteny object*

Description

An object of class Synteny.

Usage

```
data("Endosymbionts_Synteny")
```

Format

An object of class Synteny.

Details

An object of class Synteny.

Examples

```
data(Endosymbionts_Synteny)
```

EstimRearrScen *Estimate Genome Rearrangement Events with Double Cut and Join Operations*

Description

Take in a Synteny object and return predicted rearrangement events.

Usage

```
EstimRearrScen(SyntenyObject, NumRuns = -1,
               Mean = FALSE, MinBlockLength = -1,
               Verbose = TRUE)
```

Arguments

SyntenyObject	Synteny object, as obtained from running FindSynteny. Expected input is unichromosomal sequences, though multichromosomal sequences are supported.
NumRuns	Numeric; Number of times to simulate scenarios. The default value of -1 (and all non-positive values) runs each analysis for $\sqrt{b}$ iterations, where b is the number of unique breakpoints.

`Mean`	Logical; If TRUE, returns the mean number of inversions and transpositions found. If FALSE, returns the scenario corresponding to the minimum total number of operations across all runs. This parameter only affects the number of inversions and transpositions reported; the specific scenario returned is one of the runs that resulted in a minimum value.
`MinBlockLength`	Numeric; Minimum size of syntenic blocks to use for analysis. The default value accepts all blocks. Set to a larger value to ignore sections of short mutations that could be the result of SNPs or other small-scale mutations.
`Verbose`	Logical; indicates whether or not to display a progress bar and print the time difference upon completion.

Details

`EstimRearrScen` is an implementation of the Double Cut and Join (DCJ) method for analyzing large scale mutation events.

The DCJ model is commonly used to model genome rearrangement operations. Given a genome, we can create a connected graph encoding the order of conserved genomic regions. Each syntenic region is split into two nodes, with one encoding the beginning and one encoding the end (beginning and end defined relative to the direction of transcription). Each node is then connected to the two nodes it is adjacent to in the genome.

For example, given a genome with 3 syntenic regions $a - b - c$ such that b is transcribed in the opposite direction relative to a, c, our graph would consist of nodes and edges $a1 - a2 - b2 - b1 - c1 - c2$.

Given two genomes, we derive syntenic regions between the two samples and then construct two of these graph structures. A DCJ operation is one that cuts two connections of a common color and creates two new edges. The goal of the DCJ model is to rearrange the graph of the first genome into the second genome using DCJ operations. The DCJ distance is defined as the minimum number of DCJ operations to transform one graph into another.

It can be easily shown that inversions can be performed with a single DCJ operation, and block interchanges/order rearrangements can be performed with a sequence of two DCJ operations. DCJ distance defines a metric space, and prior work has demonstrated algorithms for fast computation of the DCJ distance.

However, DCJ distance inherently incentivizes inversions over block interchanges due to the former requiring half as many DCJ operations. This is a strong assumption, and there is no evidence to support gene order rearrangements occuring half as often as gene inversions.

This implementation incentivizes minimum number of total events rather than total number of DCJs. As the search space is large and multiple sequences of events can be equally parsimonious, this algorithm computes multiple scenarios with random sequences of operations to try to find the minimum amount of events. Users can choose to receive the best found solution or the mean number of events from all solutions.

Value

An *NxN* matrix of lists with the same shape as the input Synteny object. This is wrapped into a `GenRearr` object for pretty printing.

The diagonal corresponds to total sequence length of the corresponding genome.

In the upper triangle, entry `[i,j]` corresponds to the percent hits between genome `i` and genome `j`. In the lower triangle, entry `[i,j]` contains a List object with 5 properties:

- `$Inversions` and `$Transpositions` contain the (Mean/min) number of estimated inversions and transpositions (resp.) between genome `i` and genome `j`.
- `$pct_hits` contains percent hits between the genomes.
- `$Scenario` shows the sequence of events corresponding to the minimum rearrangement scenario found. See below for details.
- `$Key` provides a mapping between syntenic blocks and genome positions. See below for details.

The `print.GenRearr` method prints this data out as a matrix, with the diagonal showing the number of chromosomes and the lower triangle displaying `xI,yT`, where `x,y` the number of inversions and transpositions (resp.) between the corresponding entries.

The `$Scenario` entry describes a sequences of steps to rearrange one genome into another, as found by this algorithm. The goal of the DCJ model is to rearrange the second genome into the first. Thus, with `N` syntenic regions total, we can arbitrarily choose the syntenic blocks in genome 1 to be ordered `1,2,...,N`, and then have genome 2 numbers relative to that.

As an example, suppose genome 1 has elements `A B E(r) G` and genome 2 has elements `E B(r) A(r) G`, with X(r) denoting block X has reversed direction of transcription. We can then arbitrarily assign blocks to numbers such that genome 1 is `(1 2 3 4)` and genome 2 is `(3 -2 -1 4)`, where a negative indicates reversed direction of transcription relative to the corresponding syntenic block in genome 1.

Each entry in `$Scenario` details an operation, the result after that operation, and the number of blocks involved in the operation. If we reversed the middle two entries of genome 2, the entry in `$Scenario` would be:

```
inversion: 3 1 2 4 { 2 }
```

Here we inverted the whole block `(-2 -1)` into `(1 2)`. We could then finish the rearrangement by performing a transposition to move block 3 between 2 and 4. The entries of `$Scenario` in this case would be the following:

```
Original: 3 -2 -1 4

inversion: 3 1 2 4 { 2 }

block interchange: 1 2 3 4 { 3 }
```

Step 1 is the original state of genome 2, step 2 inverts 2 elements to arrive at `(3 1 2 4)`, and then step 3 moves one element to arrive at `(1 2 3 4)`.

It is important to note that the numbered genomic regions in `$Scenario` are not genes, they are blocks of conserved syntenic regions between the genomes. These blocks may not match up with the original blocks from the Synteny object, since some are combined during pre-processing to expedite calculations.

`$Key` is a mapping between these numbered regions and the original genomic regions. This is a 5 column matrix with the following columns (in order):

1. `start1`: Nucleotide position for the first nucleotide in of the syntenic region on genome 1.
2. `start2`: Same as `start1`, but for genome 2
3. `length`: Length of block, in nucleotides

4. `rel_direction_on_2`: 1 if the blocks have the same transcriptonal direction on both genomes, and 0 if the direction is reversed in genome 2
5. `index1`: Label of the genetic region used in `$Scenario` output

Author(s)

Aidan Lakshman (<ahl27@pitt.edu>)

References

Friedberg, R., Darling, A. E., & Yancopoulos, S. (2008). Genome rearrangement by the double cut and join operation. *Bioinformatics*, 385-416.

See Also

`FindSynteny`

`Synteny`

Examples

```
db <- system.file("extdata", "Influenza.sqlite", package="DECIPHER")
synteny <- FindSynteny(db)
synteny

rearrs <- EstimRearrScen(synteny)

rearrs          # view whole object
rearrs[[2,1]]   # view details on Genomes 1 and 2
```

ExampleStreptomycesData

Example ProtWeaver Input Data from Streptomyces *Species*

Description

Data from Streptomyces species to test `ProtWeaver` functionality.

Usage

```
data("ExampleStreptomycesData")
```

Format

The data contain two elements, `Genes` and `Tree`. `Genes` is a list of presence/absence vectors in the input required for `ProtWeaver`. `Tree` is a species tree used for additional input.

Details

This dataset contains a number of Clusters of Orthologous Genes (COGs) and a species tree for use with ProtWeaver. This dataset showcases an example of using ProtWeaver with a list of vectors. Entries in each vector are formatted correctly for use with co-localization prediction. Each COG `i` contains entries of the form `a_b_c`, indicating that the gene was found in genome `a` on chromosome `b`, and was at the `c`'th location. The original dataset is comprised of 301 unique genomes.

See Also

ProtWeaver

Examples

```
exData <- get(data("ExampleStreptomycesData"))
pw <- ProtWeaver(exData$Genes)
# Subset isn't necessary but is faster for a working example
predict(pw, Subset=1:10, MySpeciesTree=exData$Tree)
```

ExtractBy *Extract and organize* DNAStringSets.

Description

Return organized `DNAStringSets` based on three currently supported object combinations. First return a single `DNAStringSet` of feature sequences from a DFrame of genecalls and a `DNAStingSet` of the source assembly. Second return a list of `DNAStringSets` of predicted pairs from a `PairSummaries` object and a character string of the location of a `DECIPHER` SQLite database. Third return a list of `DNAStringSets` of predicted single linkage communities from a `PairSummaries` object, a character string of the location of a `DECIPHER` SQLite database, and a list of identifiers generated by `DisjointSet`.

Usage

```
ExtractBy(x,
          y,
          z,
          Verbose = FALSE)
```

Arguments

`x`	A `PairSummaries` object, or if `y` is a `DNAStringSet`, a `DFrame` of gene calls such as one generated by `gffToDataFrame`.
`y`	A character vector of length 1 indicating the location of a `DECIPHER` SQLite database. Or, if `x` is a `DFrame`, a `DNAStringSet` of the assembly the gene calls are called from.
`z`	Optional; a list of identifiers generated by `DisjointSet`. Or any list built along a similar format with identifiers paired to the PairSummaries object.

Verbose — Logical indicating whether to print progress bars and messages. Defaults to FALSE.

Details

All sequences are forced into the same direction based on the Strand column supplied by either the gene calls DFrame specified by x, or the GeneCalls attribute of the PairSummaries object specified by y.

Value

Return a DNAStringSet, or list of DNAStringSets arranged depending upon the objects supplied. See description.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class, PairSummaries, DisjointSet

Examples

```
DBPATH <- system.file("extdata",
                      "Endosymbionts.sqlite",
                      package = "SynExtend")
data("Endosymbionts_Pairs03", package = "SynExtend")
data("Endosymbionts_Sets", package = "SynExtend")

# extract the first 10 disjoint sets
Sets <- ExtractBy(x = Endosymbionts_Pairs03,
                  y = DBPATH,
                  z = Endosymbionts_Sets[1:10],
                  Verbose = TRUE)



# extract just the pairs
Sets <- ExtractBy(x = Endosymbionts_Pairs03,
                  y = DBPATH,
                  Verbose = TRUE)
```

FindSets *Find all single linkage clusters in an undirected pairs list.*

Description

Take in a pair of vectors representing the columns of an undirected pairs list and return the single linkage clusters.

Usage

```
FindSets(p1,
         p2,
         Verbose = FALSE)
```

Arguments

p1	Column 1 of a pairs matrix or list.
p2	Column 2 of a pairs matrix or list.
Verbose	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

FindSets uses a version of the union-find algorithm to collect single linkage clusters from a pairs list. Currently meant to be used inside a wrapper function, but left exposed for user convenience.

Value

A two column matrix with the first column being input nodes, and the second the node representing a single linkage cluster.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

PairSummaries

Examples

```
set.seed(1986)
m <- cbind(as.integer(sample(30, size = 25,
                             replace = TRUE)),
           as.integer(sample(35, size = 25,
                             replace = TRUE)))

Levs <- unique(c(m[, 1],
```

```
                  m[, 2]))
m <- cbind("1" = as.integer(factor(x = m[, 1L],
                                   levels = Levs)),
           "2" = as.integer(factor(x = m[, 2L],
                                   levels = Levs)))
z <- FindSets(p1 = m[, 1],
              p2 = m[, 2])
```

`Generic` *Model for predicting PID based on k-mer statistics*

Description

Though the function PairSummaries provides an argument allowing users to ask for alignments, given the time consuming nature of that process on large data, models are provided for predicting PIDs of pairs based on k-mer statistics without performing alignments.

Usage

```
data("Generic")
```

Format

The format is an object of class "glm".

Details

A model for predicting the PID of a pair of sequences based on the k-mers that were used to link the pair.

Examples

```
data(Generic)
```

`GetProtWebData.ProtWeb`
Extract information from a ProtWeb object

Description

ProtWeb objects are outputted from `predict.ProtWeaver`.

This function extracts the underlying data from the object.

Usage

```
## S3 method for class 'ProtWeb'
GetProtWebData(x, AsDf=FALSE, ...)
```

Arguments

x	A ProtWeb object
AsDf	Should data be printed as a pairwise entry? If TRUE, returns a matrix with three columns, where the first two columns define the pair of genes/proteins and the third column defines the prediction/score. If FALSE, returns an adjacency matrix encoding the same information (but may be sparse depending on how many predictions were made in the original predict call.)
...	Additional parameters for consistency with generic.

Details

predict.ProtWeaver returns a ProtWeb object, which bundles some methods to make formatting and printing of results slightly nicer. This method extracts data form the ProtWeb object.

If AsDf=TRUE, the return data is a 3xN data.frame, with columns 3 showing the prediction for the pair of genes/proteins specified in columns 1 and 2. This format is when predictions are made on small number of pairs (meaning the resulting adjacency matrix is sparse).

If AsDf=FALSE, the return data is a NxN adjacency matrix, with entry i,j containing the prediction for genes i and j.

Value

Either a data.frame or a matrix.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

See Also

predict.ProtWeaver

Examples

```
###############
## Prediction with built-in model and data
###############

exData <- get(data("ExampleStreptomycesData"))

# Subset isn't necessary but is faster for a working example
pw <- ProtWeaver(exData$Genes[1:10])

protweb <- predict(pw, method='Jaccard')

# print out results as an adjacency matrix
GetProtWebData(protweb)

# print out results as a pairwise data.frame
GetProtWebData(protweb, AsDf=TRUE)
```

gffToDataFrame *Generate a DataFrame of gene calls from a gff3 file*

Description

Generate a DataFrame of gene calls from a gff3 file

Usage

```
gffToDataFrame(GFF,
               AdditionalAttrs = NULL,
               AdditionalTypes = NULL,
               RawTableOnly = FALSE,
               Verbose = FALSE)
```

Arguments

GFF	A url or filepath specifying a gff3 file to import
AdditionalAttrs	A vector of character strings to designate the attributes to pull. Default Attributes include: "ID", "Parent", "Name", "gbkey", "gene", "product", "protein_id", "gene_biotype", "transl_table", and "Note".
AdditionalTypes	A vector of character strings to query from the the "Types" column. Default types are limited to "Gene" and "Pseudogene", but any possible entry for "Type" in a gff3 format can be added, such as "rRNA", or "CRISPR_REPEAT".
RawTableOnly	Logical specifying whether to return the raw imported GFF without complex parsing. Remains as a holdover from function construction and debugging. For simple gff3 import see `rtracklayer::import`.
Verbose	Logical specifying whether to print a progress bar and time difference.

Details

Import a gff file into a rectangular parsable object.

Value

A DataFrame with relevant information extracted from a GFF.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

Examples

```
ImportedGFF <- gffToDataFrame(GFF = system.file("extdata",
                                          "GCF_021065005.1_ASM2106500v1_genomic.gff.gz",
                                                package = "SynExtend"),
                              Verbose = TRUE)
```

`LinkedPairs` *Tables of where syntenic hits link pairs of genes*

Description

Syntenic blocks describe where order is shared between two sequences. These blocks are made up of exact match hits. These hits can be overlayed on the locations of sequence features to clearly illustrate where exact sequence similarity is shared between pairs of sequence features.

Usage

```
## S3 method for class 'LinkedPairs'
print(x,
      quote = FALSE,
      right = TRUE,
      ...)
```

Arguments

`x`	An object of class `LinkedPairs`.
`quote`	Logical indicating whether to print the output surrounded by quotes.
`right`	Logical specifying whether to right align strings.
`...`	Other arguments for `print`.

Details

Objects of class `LinkedPairs` are stored as square matrices of list elements with `dimnames` derived from the `dimnames` of the object of class "Synteny" from which it was created. The diagonal of the matrix is only filled if `OutputFormat` "Comprehensive" is selected in `NucleotideOverlap`, in which case it will be filled with the gene locations supplied to `GeneCalls`. The upper triangle is always filled, and contains location information in nucleotide space for all syntenic hits that link features between sequences in the form of an integer matrix with named columns. "QueryGene" and "SubjectGene" correspond to the integer rownames of the supplied gene calls. "QueryIndex" and "SubjectIndex" correspond to "Index1" and "Index2" columns of the source synteny object position. Remaining columns describe the exact positioning and size of extracted hits. The lower triangle is not filled if `OutputFormat` "Sparse" is selected and contains relative displacement positions for the 'left-most' and 'right-most' hit involved in linking the particular features indicated in the related line up the corresponding position in the upper triangle.

The object serves only as a simple package for input data to the `PairSummaries` function, and as such may not be entirely user friendly. However it has been left exposed to the user should they find this data interesting.

Value

An object of class "LinkedPairs".

Author(s)

Nicholas Cooley <npc19@pitt.edu>

`NucleotideOverlap` *Tabulating Pairs of Genomic Sequences*

Description

A function for concisely tabulating where genomic features are connected by syntenic hits.

Usage

```
NucleotideOverlap(SyntenyObject,
                  GeneCalls,
                  LimitIndex = FALSE,
                  AcceptContigNames = TRUE,
                  Verbose = FALSE)
```

Arguments

`SyntenyObject`	An object of class "Synteny" built from the `FindSynteny` in the package `DECIPHER`.
`GeneCalls`	A named list of objects of class "DFrame" built from `gffToDataFrame`, objects of class "GRanges" imported from `rtracklayer::import`, or objects of class "Genes" created from the `DECIPHER` function `FindGenes`. "DFrame"s built by "gffToDataFrame" can be used directly, while "GRanges" objects may also be used with limited functionality. Using a "GRanges" object will force all alignments to nucleotide alignments. Objects of class "Genes" generated by `FindGenes` function equivalently to those produced by `gffToDataFrame`. Using a "GRanges" object will force `LimitIndex` to `TRUE`.
`LimitIndex`	Logical indicating whether to limit which indices in a synteny object to query. `FALSE` by default, when `TRUE` only the first sequence in all selected identifiers will be used. `LimitIndex` can be used to skip analysis of plasmids, or solely query a single chromosome.
`AcceptContigNames`	Match names of contigs between gene calls object and synteny object. Where relevant, the first white space and everything following are removed from contig names. If "TRUE", NucleotideOverlap assumes that the contigs at each position in the synteny object and "GeneCalls" object are in the same order. Is automatically set to TRUE when "GeneCalls" are of class "GRanges".
`Verbose`	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

Builds a matrix of lists that contain information about linked pairs of genomic features.

Value

An object of class "LinkedPairs". "LinkedPairs" is fundamentally just a list in the form of a matrix. The lower triangle of the matrix is populated with matrices that contain all kmer hits from the "Synteny" object that link features from the "GeneCalls" object. The upper triangle is populated by matrices of the summaries of those hits by feature. The diagonal is populated by named vectors of the lengths of the contigs, much like in the "Synteny" object. The "LinkedPairs" object also contains a "GeneCalls" attribute that contains the user supplied features in a slightly more trimmed down form. This allows users to only need to supply gene calls once and not again in the "PairSummaries" function.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class

Examples

```
data("Endosymbionts_GeneCalls", package = "SynExtend")
data("Endosymbionts_Synteny", package = "SynExtend")

Links <- NucleotideOverlap(SyntenyObject = Endosymbionts_Synteny,
                           GeneCalls = Endosymbionts_GeneCalls,
                           LimitIndex = FALSE,
                           Verbose = TRUE)
```

PairSummaries *Summarize connected pairs in a LinkedPairs object*

Description

Takes in a "LinkedPairs" object and gene calls, and returns a data.frame of paired features.

Usage

```
PairSummaries(SyntenyLinks,
              DBPATH,
              PIDs = FALSE,
              Score = FALSE,
              IgnoreDefaultStringSet = FALSE,
              Verbose = FALSE,
              Model = "Generic",
```

```
                DefaultTranslationTable = "11",
                AcceptContigNames = TRUE,
                OffSetsAllowed = NULL,
                Storage = 1,
                ...)
```

Arguments

SyntenyLinks	A LinkedPairs object. In previous versions of this function, a GeneCalls object was also required, but this object is now carried forward from NucleotideOverlap inside the LinkedPairs object.
DBPATH	A SQLite connection object or a character string specifying the path to the database file constructed from DECIPHER's Seqs2DB function. This path is always required as "PairsSummaries" always computes the tetramer distance between paired sequences.
PIDs	Logical indicating whether to provide a PID for each pair. If TRUE all pairs will be aligned using DECIPHER's AlignProfiles. This step can be time consuming, especially for large numbers of pairs. Default is FALSE.
Score	Logical indicating whether to provide a length normalized score with DECIPHER's ScoreAlignment function. If TRUE all pairs will be aligned using DECIPHER's AlignProfiles. This step can be time consuming, especially for large numbers of pairs. Default is FALSE.
IgnoreDefaultStringSet	Logical indicating alignment type preferences. If FALSE (the default) pairs that can be aligned in amino acid space will be aligned as an AAStringSet. If TRUE all pairs will be aligned in nucleotide space. For PairSummaries to align the translation of a pair of sequences, both sequences must be tagged as coding in the "GeneCalls" object, and be the correct width for translation.
Verbose	Logical indicating whether or not to display a progress bar and print the time difference upon completion.
Model	A character string specifying a model to use to predict PIDs without performing an alignment. By default this argument is "Generic" specifying a generic PID prediction model based on PIDs computed from a randomly selected set of genomes. Currently no other models are included. Users may also supply their own model of type "glm" if they so desire in the form of an RData file. This model will need to take in some, or of the columns of statistics per pair that PairSummaries supplies.
DefaultTranslationTable	A character used to set the default translation table for translate. Is passed to getGeneticCode. Used when no translation table is specified in the "GeneCalls" object.
AcceptContigNames	Match names of contigs between gene calls object and synteny object. Where relevant, the first white space and everything following are removed from contig names. If TRUE, PairSummaries assumes that the contigs at each position in the synteny object and "GeneCalls" object are in the same order. Is automatically

set to TRUE when "GeneCalls" are of class "GRanges". Is currently TRUE by default.

OffSetsAllowed Defaults to NULL. Supplying an integer vector will indicate gap sizes to attempt to fill. A value of 2 will attempt to span gaps of size 1. If a vector larger than 1 is provided, i.e. c(2, 3), will attempt to query all gap sizes implied by the vector, in this case gaps of size 1 and 2.

Storage Numeric indicating the approximate size a user wishes to allow for holding StringSets in memory to extract gene sequences, in "Gigabytes". The lower Storage is set, the more likely that PairSummaries will need to reaccess StringSets when extracting gene sequences. The higher Storage is set, the more sequences PairSummaries will attempt to hold in memory, avoiding the need to re-access the source database many times. Set to 1 by default, indicating that PairSummaries can store a "Gigabyte" of sequences in memory at a time.

... Arguments to be passed to AlignProfiles, and DistanceMatrix.

Details

The LinkedPairs object generated by NucleotideOverlap is a container for raw data that describes possible orthologous relationships, however ultimate assignment of orthology is up to user discretion. PairSummaries generates a clear table with relevant statistics for a user to work with as they choose. The option to align all pairs, though onerous can allow users to apply a hard threshold to predictions by PID, while built in models can allow more expedient thresholding from predicted PIDs.

Value

A data.frame of class "data.frame" and "PairSummaries" of paired genes that are connected by syntenic hits. Contains columns describing the k-mers that link the pair. Columns "p1" and "p2" give the location ids of the the genes in the pair in the form "DatabaseIdentifier_ContigIdentifier_GeneIdentifier". "ExactMatch" provides an integer representing the exact number of nucleotides contained in the linking k-mers. "TotalKmers" provides an integer describing the number of distinct k-mers linking the pair. "MaxKmer" provides an integer describing the largest k-mer that links the pair. A column titled "Consensus" provides a value between zero and 1 indicating whether the kmers that link a pair of features are in the same position in each feature, with 1 indicating they are in exactly the same position and 0 indicating they are in as different a position as is possible. The "Adjacent" column provides an integer value ranging between 0 and 2 denoting whether a feature pair's direct neighbors are also paired. Gap filled pairs neither have neighbors, or are included as neighbors. The "TetDist" column provides the euclidean distance between oligonucleotide - of size 4 - frequences between predicted pairs. "PIDType" provides a character vector with values of "NT" where either of the pair indicates it is not a translatable sequence or "AA" where both sequences are translatable. If users choose to perform pairwise alignments there will be a "PID" column providing a numeric describing the percent identity between the two sequences. If users choose to predict PIDs using their own, or a provided model, a "PredictedPID" column will be provided.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class, NucleotideOverlap

Examples

```
DBPATH <- system.file("extdata",
                      "Endosymbionts.sqlite",
                      package = "SynExtend")

data("Endosymbionts_LinkedFeatures", package = "SynExtend")

Pairs <- PairSummaries(SyntenyLinks = Endosymbionts_LinkedFeatures,
                       PIDs = FALSE,
                       DBPATH = DBPATH,
                       Verbose = TRUE)
```

---

plot.ProtWeb *Plot predictions in a ProtWeb object*

Description

ProtWeb objects are outputted from predict.ProtWeaver.

This function plots the predictions in the object using a force-directed embedding of connections in the adjacency matrix.

This function is still a work in progress.

Usage

```
## S3 method for class 'ProtWeb'
plot(x, NumSims=10,
      Gravity=0.05, Coulomb=0.1, Connection=5,
      MoveRate=0.25, Cutoff=0.2, ColorPalette=topo.colors,
      Verbose=TRUE, ...)
```

Arguments

x	A ProtWeb object. See predict.ProtWeaver
NumSims	Number of iterations to run the model for.
Gravity	Strength of Gravity force. See 'Details'.
Coulomb	Strength of Coulomb force. See 'Details'.
Connection	Strength of Connective force. See 'Details'.
MoveRate	Controls how far each point moves in each iteration.
Cutoff	Cutoff value; if abs(val) < Cutoff, that Connection is shrunk to zero.

ColorPalette	Color palette for graphing. Valid inputs are any palette available in palette.pals(). See palette for more info.
Verbose	Logical indicating whether to print progress bars and messages. Defaults to TRUE.
...	Additional parameters for consistency with generic.

Details

This function plots the ProtWeb object using a force-directed embedding. This embedding has three force components:

- Gravity Force: Attractive force pulling nodes towards (0,0)
- Coulomb Force: Repulsive force pushing close nodes away from each other
- Connective Force: Tries to push node connections to equal corresponding values in the adjacency matrix

The parameters in the function are sufficient to get an embedding, though users are welcome to try to tune them for a better visualization. This function is meant to aid with visualization of the adjacency matrix, not for concrete analyses of clusters.

The function included in this release is early stage. Next release cycle will update this function with an updated version of this algorithm to improve plotting, visualization, and runtime.

Value

No return value; creates a plot in the graphics window.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

See Also

predict.ProtWeaver

GetProtWebData.ProtWeb

Examples

```
exData <- get(data("ExampleStreptomycesData"))
pw <- ProtWeaver(exData$Genes)

# Subset isn't necessary but is faster for a working example
# Same w/ method='Jaccard'
protweb <- predict(pw, 'Jaccard', subset=1:50)

plot(protweb)
```

`predict.ProtWeaver` *Make predictions with ProtWeaver objects*

Description

This S3 method predicts a functional association network from a ProtWeaver object. This returns an object of type `ProtWeb`, which is essentially an adjacency matrix with some extra S3 methods to make printing cleaner.

Usage

```
## S3 method for class 'ProtWeaver'
predict(object, Method='Ensemble',
         Subset=NULL, NumCores=1,
         MySpeciesTree=NULL, PretrainedModel=NULL,
         RawZScores=FALSE, NoPrediction=FALSE,
         ReturnRawData=FALSE, Verbose=TRUE, ...)
```

Arguments

`object`	A ProtWeaver object
`Method`	Method to use for prediction. See 'Details'.
`Subset`	Subset of data to predict on. This can either be a vector or a `2xN` matrix. If a vector, prediction proceeds for all possible pairs of elements specified in the vector (either by name, for character vector, or by index, for numeric vector). For example, `subset=1:3` will predict for pairs `(1,2)`, `(1,3)`, `(2,3)`. If a matrix, subset is interpreted as a matrix of pairs, where each row of the matrix specifies a pair to evaluate. These can also be specifed by name (character) or by index (numeric). `subset=cbind(c(1,1,2), c(2,3,3))` produces equivalent functionality to `subset=1:3`.
`NumCores`	Number of cores to use for methods that support multithreaded execution. Currently only supported for methods `'ProfDCA'` and `'Ensemble'`. Setting to a negative value will use one less than the value of `detectCores()`, or one core if the number of available cores cannot be determined. See Note for more information. This parameter has no effect on Windows due to reliance on forking via `mclapply`.
`MySpeciesTree`	Phylogenetic tree of all genomes in the dataset. Required for `Method='Behdenna'`, and can improve predictions for other methods. `'Behdenna'` requires a rooted, bifurcating tree (other values of `Method` can handle arbitrary trees).
`PretrainedModel`	A pretrained model for use with ensemble predictions. If unspecified when `Method='Ensemble'`, the program will use built-in models (see `BuiltInEnsembles`). See the examples for how to train an ensemble method to pass to `PretrainedModel`. Has no effect if `Method != 'Ensemble'`.

RawZScores	For methods that return z-scores, should raw scores be returned? If FALSE, instead returns normalized absolute value of predictions. These tend to be better predictions. Currently, only Method='Behdenna' uses this parameter.
NoPrediction	For Method='Ensemble', should data be returned prior to making predictions? If TRUE, this will instead return a dataframe (data.frame) with predictions from each algorithm for each pair. This dataframe is typically used to train an ensemble model. If FALSE, ProtWeaver will return predictions for each pair (using user model if provided or a built-in otherwise).
ReturnRawData	Internal parameter used for ensemble predictions. If TRUE, returns predictions without formatting them into a ProtWeb object. Users should specify NoPrediction=TRUE rather than use this parameter (see Details).
Verbose	Logical indicating whether to print progress bars and messages. Defaults to TRUE.
...	Additional parameters for other predictors and consistency with generic.

Details

predict.ProtWeaver wraps several methods to create an easy interface for multiple prediction types. The following values of Method are currently supported:

- 'Jaccard': Jaccard distance of PA profiles
- 'Hamming': Hamming distance of PA profiles
- 'MutualInformation': MI of PA profiles
- 'ProfDCA': Direct Coupling Analysis of PA profiles
- 'Behdenna': Analysis of Gain/Loss events following Behdenna et al. (2016)
- 'Coloc': Co-localization analysis
- 'MirrorTree': MirrorTree
- 'ContextTree': ContextTree

(PA = Presence/Absence)

This returns a ProtWeb object, an S3 class that makes formatting and printing of results slightly nicer. Data can be extracted form the ProtWeb object with:

```
GetProtWebData(ProtWebObject, AsDf=c(T,F))
```

Different methods require different types of input. The constructor ProtWeaver will notify the user which methods are runnable with the given data. Note that method Behdenna requires a species tree, which must be bifurcating. Method Ensemble automatically selects the methods that can be run with the given input data.

See ProtWeaver for more information on input data types.

Value

Returns a ProtWeb object. See GetProtWebData for more info.

Note

Note that the pairwise associations are stored in a matrix, meaning that if the ProtWeaver object contains 100 entries, the output ProtWeb object contains a `100x100` matrix. Users should be advised that predicting too many pairs can lead to vector memory exhaustion errors. On my machine, ProtWeaver supports predictions on up to around $53,000^2$ pairs (meaning all possible pairs between 53,000 proteins/genes). Next release will add more memory efficient storage to further increase this limit, though the memory capacity can never be better than $O(P^2)$, with `P` the number of proteins/genes.

`NumCores` only uses 1 less core than is detected, or 1 core if `detectCores()` cannot detect the number of available cores. This is because of a recurring issue on my machine where the R session takes all available cores and is then locked out of forking processes, with the only solution to restart the entire R session. This may be an issue specific to ARM Macs, but out of an abundance of caution I've made the default setting to be slightly slower but guarantee completion rather than risk bricking a machine.

More models will be implemented in the future. Planned models for next release include:

- Random Forests for Ensemble predictions
- XGBoost for Ensemble predictions
- Normalized Phylogenetic Profiles
- SVDPhy
- DCA at the residue level (Weigt et al. 2009)

Feel free to contact me regarding other models you would like to see added.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

References

Behdenna, A., et al., *Testing for Independence between Evolutionary Processes.* Systematic Biology, 2016. **65**(5): p. 812-823.

Franceschini, A., et al., *SVD-phy: improved prediction of protein functional associations through singular value decomposition of phylogenetic profiles.* Bioinformatics, 2016. **32**(7): p. 1085-1087.

Fukunaga, T. and W. Iwasaki, *Inverse Potts model improves accuracy of phylogenetic profiling.* Bioinformatics, 2022.

Lokhov, A.Y., et al., *Optimal structure and parameter learning of Ising models.* Science advances, 2018. **4**(3): p. e1700791.

Pazos, F. and A. Valencia, *Similarity of phylogenetic trees as indicator of protein–protein interaction.* Protein Engineering, Design and Selection, 2001. **14**(9): p. 609-614.

Pazos, F., et al., *Assessing protein co-evolution in the context of the tree of life assists in the prediction of the interactome.* J Mol Biol, 2005. **352**(4): p. 1002-15.

Sadreyev, I.R., et al., *PhyloGene server for identification and visualization of co-evolving proteins using normalized phylogenetic profiles.* Nucleic Acids Research, 2015. **43**(W1): p. W154-W159.

Sato, T., et al., *The inference of protein-protein interactions by co-evolutionary analysis is improved by excluding the information about the phylogenetic relationships.* Bioinformatics, 2005. **21**(17): p. 3482-9.

Sato, T., et al., *Partial correlation coefficient between distance matrices as a new indicator of protein-protein interactions.* Bioinformatics, 2006. **22**(20): p. 2488-92.

Weigt, M., et al., *Identification of direct residue contacts in protein-protein interaction by message passing.* Proceedings of the National Academy of Sciences, 2009. **106**(1): p. 67-72.

See Also

ProtWeaver

GetProtWebData

Examples

```
##############
## Prediction with built-in model and data
###############

exData <- get(data("ExampleStreptomycesData"))
pw <- ProtWeaver(exData$Genes[1:50])

# Subset isn't necessary but is faster for a working example
protweb1 <- predict(pw, Subset=1:10, MySpeciesTree=exData$Tree)

# print out results as an adjacency matrix
GetProtWebData(protweb1)

###############
## Training own ensemble model
###############

datavals <- predict(pw, NoPrediction=TRUE)

actual_values <- sample(c(0,1), nrow(datavals), replace=TRUE)
# This example just picks random numbers
# ***Do not do this for your own models***

# Make sure the actual values correspond to the right pairs!
datavals[,'y'] <- actual_values
myModel <- glm(y~., datavals[,-c(1,2)], family='binomial')

testProtWeaverObject <- ProtWeaver(exData$Genes[51:60])
protweb2 <- predict(testProtWeaverObject,
                    PretrainedModel=myModel)

# Print result as a 3xN matrix of pairwise scores
GetProtWebData(protweb2, AsDf=TRUE)
```

ProtWeaver *ProtWeaver: Predicting Protein Functional Association Networks*

Description

ProtWeaver is an S3 class with methods for predicting functional association using protein or gene data. ProtWeaver implements several methods utilized in the literature, with many more planned for future implementation. For details on predictions, see `predict.ProtWeaver`.

Usage

```
ProtWeaver(ListOfData, NoWarn=FALSE)
```

Arguments

`ListOfData`	A list of gene data, where each entry corresponds to information on a particular gene. List must contain either dendrograms or vectors, and cannot contain a mixture. If list is composed of dendrograms, each dendrogram is a gene tree for the corresponding entry. If list is composed of vectors, vectors should be numeric or character vectors denoting the genomes containing that gene.
`NoWarn`	Several algorithms depend on having certain data. When a `ProtWeaver` object is initialized, it automatically selects which algorithms can be used given the input data. By default, ProtWeaver will notify the user of algorithms that cannot be used with warnings. Setting `NoWarn=TRUE` will suppress these messages.

Details

ProtWeaver expects input data to be a list. All entries must be one of the following:

1. `ListOfData[[i]] = c('ID#1', 'ID#2', ..., 'ID#k')`
2. `ListOfData[[i]] = c('i1_d1_p1', 'i2_d2_p2', ..., 'ik_dk_pk')`
3. `ListOfData[[i]] = dendrogram(...)`

In (1), each ID#i corresponds to the unique identifier for genome #i. For entry #j in the list, the presence of 'ID#i' means genome #i has an ortholog for gene/protein #j.

Case (2) is the same as (1), just with the formatting of names slightly different. Each entry is of the form `i_d_p`, where `i` is the unique identifier for the genome, `d` is which chromosome the ortholog is located, and `p` is what position the ortholog appears in on that chromosome. `p` must be a `numeric`, while the other entries can be any value.

Case (3) expects gene trees for each gene, with labeled leaves corresponding to each source genome. If `ListOfData` is in this format, taking `labels(ListOfData[[i]])` should produce a character vector that matches the format of either (2) or (1).

See the Examples section for illustrative examples.

ProtWeaver requires input of scenario (3) to use MirrorTree or ContextTree, and requires input of scenario (2) (or (3) with leaves labeled according to (2)) for co-localization analyses.

Note that ALL entries must belong to the same category–a combination of character vectors and dendrograms is not allowed.

Prediction of a functional association network is done using `predict(ProtWeaverObject)`. See `predict.ProtWeaver` for more information.

Value

Returns a ProtWeaver object.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

See Also

`predict.ProtWeaver`, `ExampleStreptomycesData`, `BuiltInEnsembles`

Examples

```
# I'm using gene to mean either a gene or protein

## Imagine we have the following 4 genomes:
## (each letter denotes a distinct gene)
##    Genome 1: a b c d
##    Genome 2: d c e
##    Genome 3: b a e
##    Genome 4: a e

## We have 5 total genes: (a,b,c,d,e)
##    a is present in genomes 1, 3, 4
##    b is present in genomes 1, 3
##    c is present in genomes 1, 2
##    d is present in genomes 1, 2
##    e is present in genomes 2, 3, 4

## Constructing a ProtWeaver object according to (1):
l <- list()
l[['a']] <- c('1', '3', '4')
l[['b']] <- c('1', '3')
l[['c']] <- c('1', '2')
l[['d']] <- c('1', '2')
l[['e']] <- c('2', '3', '4')

## Each value of the list corresponds to a gene
## The associated vector shows which genomes have that gene
pwCase1 <- ProtWeaver(l)

## Constructing a ProtWeaver object according to (2):
##  Here we need to add in the chromosome and the position
##  As we only have one chromosome,
##  we can just set that to 1 for all.
##  Position can be identified with knowledge, or with
```

```
##  FindGenes(...) from DECIPHER.

## In this toy case, genomes are small so it's simple.
l <- list()
l[['a']] <- c('1_1_1', '3_1_2', '4_1_1')
l[['b']] <- c('1_1_2', '3_1_1')
l[['c']] <- c('1_1_3', '2_1_2')
l[['d']] <- c('1_1_4', '2_1_1')
l[['e']] <- c('2_1_3', '3_1_3', '4_1_2')

pwCase2 <- ProtWeaver(l)


## For Case 3, we just need dendrogram objects for each
# l[['a']] <- dendrogram(...)
# l[['b']] <- dendrogram(...)
# l[['c']] <- dendrogram(...)
# l[['d']] <- dendrogram(...)
# l[['e']] <- dendrogram(...)

## Leaf labels for these will be the same as the
##  entries in Case 1.
```

SequenceSimilarity *Return a numeric value that represents the similarity between two aligned sequences as determined by a provided subsitution matrix.*

Description

Takes in a DNAStringSet or AAStringSet representing a pairwise alignment and a subsitution matrix such as those present in PFASUM, and return a numeric value representing sequence similarity as defined by the substitution matrix.

Usage

```
SequenceSimilarity(Seqs,
                   SubMat,
                   penalizeGapLetter = TRUE,
                   includeTerminalGaps = TRUE,
                   allowNegative = TRUE)
```

Arguments

Seqs	A DNAStringSet or AAStringSet of length 2.
SubMat	A named matrix representing a substitution matrix. If left "NULL" and "Seqs" is a AAStringSet, the 40th "PFASUM" matrix is used. If left "NULL" and "Seqs" is a DNAStringSet, a matrix with only the diagonal filled with "1"'s is used.

penalizeGapLetter
: A logical indicating whether or not to penalize Gap-Letter matches. Defaults to "TRUE".

includeTerminalGaps
: A logical indicating whether or not to penalize terminal matches. Defaults to "TRUE".

allowNegative
: A logical indicating whether or not allow negative scores. Defaults to "TRUE". If "FALSE" scores that are returned as less than zero are converted to zero.

Details

Takes in a DNAStringSet or AAStringSet representing a pairwise alignment and a subsitution matrix such as those present in PFASUM, and return a numeric value representing sequence similarity as defined by the substitution matrix.

Value

Returns a single numeric.

Author(s)

Erik Wright <ESWRIGHT@pitt.edu> Nicholas Cooley <npc19@pitt.edu>

See Also

AlignSeqs, AlignProfiles, AlignTranslation, DistanceMatrix

Examples

```
db <- system.file("extdata", "Bacteria_175seqs.sqlite", package = "DECIPHER")
dna <- SearchDB(db, remove = "all")
alignedDNA <- AlignSeqs(dna[1:2])

DNAPlaceholder <- diag(15)
dimnames(DNAPlaceholder) <- list(DNA_ALPHABET[1:15],
                                 DNA_ALPHABET[1:15])

SequenceSimilarity(Seqs = alignedDNA,
                   SubMat = DNAPlaceholder,
                   includeTerminalGaps = TRUE,
                   penalizeGapLetter = TRUE,
                   allowNegative = TRUE)
```

SubSetPairs *Subset a "PairSummaries" object.*

Description

For a given object of class "PairSummaries", pairs based on either competing predictions, user thresholds on prediction statistics, or both.

Usage

```
SubSetPairs(CurrentPairs,
            UserThresholds,
            RejectCompetitors = TRUE,
            RejectionCriteria = "PID",
            WinnersOnly = TRUE,
            Verbose = FALSE)
```

Arguments

CurrentPairs	An object of class "PairSummaries". Can also take in a generic "data.frame", as long as the feature naming scheme is the same as that followed by all SynExtend functions.
UserThresholds	A named vector where values indicate a threshold for statistics to be above, and names designate which statistic to threshold on.
RejectCompetitors	A logical that defaults to "TRUE". Allowing users to choose to remove competing predictions. When set to "FALSE", no competitor rejection is performed. When "TRUE" all competing pairs with the exception of the best pair as determined by "RejectionCriteria" are rejected. Can additionally be set to a numeric or integer, in which case only competing predictions below that value are dropped.
RejectionCriteria	A character indicating which column value competitor rejection should reference. Defaults to "PID".
WinnersOnly	A logical indicating whether or not to return just the pairs that are selected. Defaults to "TRUE" to return a subset object of class "PairSummaries". When "FALSE", function returns a list of two "PairSummaries" objects, one of the selected pairs, and the second of the rejected pairs.
Verbose	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

SubSetPairs uses a naive competitor rejection algorithm to remove predicted pairs when nodes are predicted to be paired to multiple nodes within the same index.

Value

An object of class "PairSummaries", or a list of two "PairSummaries" objects.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

PairSummaries NucleotideOverlap

Examples

```
data("Endosymbionts_Pairs03", package = "SynExtend")
# remove competitors under default conditions
Pairs2 <- SubSetPairs(CurrentPairs = Endosymbionts_Pairs03,
                      Verbose = TRUE)
THRESH <- c(0.5, 21)
names(THRESH) <- c("Consensus", "ExactMatch")
# remove pairs only based on user defined thresholds
Pairs3 <- SubSetPairs(CurrentPairs = Endosymbionts_Pairs03,
                      UserThresholds = THRESH,
                      RejectCompetitors = FALSE,
                      Verbose = TRUE)
```

Index