Package ‘SynExtend’

May 30, 2024

Type Package

Title Tools for Working With Synteny Objects

Version 1.16.0

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 (>= 4.3.0), DECIPHER (>= 2.28.0)

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

Suggests BiocStyle, knitr, igraph, markdown, rmarkdown

License GPL-3

ByteCompile true

Encoding UTF-8

NeedsCompilation yes

VignetteBuilder knitr

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

git_branch RELEASE_3_19

git_last_commit 4426e94

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-29

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>
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlastSeqs</td>
<td>3</td>
</tr>
<tr>
<td>BlockExpansion</td>
<td>4</td>
</tr>
<tr>
<td>BlockReconciliation</td>
<td>6</td>
</tr>
<tr>
<td>BuiltInEnsembles</td>
<td>8</td>
</tr>
<tr>
<td>CIDist_NullDist</td>
<td>9</td>
</tr>
<tr>
<td>ClusterByK</td>
<td>10</td>
</tr>
<tr>
<td>dendrapply</td>
<td>11</td>
</tr>
<tr>
<td>DisjointSet</td>
<td>14</td>
</tr>
<tr>
<td>DPhyloStatistic</td>
<td>15</td>
</tr>
<tr>
<td>Endosymbionts_GeneCalls</td>
<td>17</td>
</tr>
<tr>
<td>Endosymbionts_LinkedFeatures</td>
<td>17</td>
</tr>
<tr>
<td>Endosymbionts_Pairs01</td>
<td>18</td>
</tr>
<tr>
<td>Endosymbionts_Pairs02</td>
<td>18</td>
</tr>
<tr>
<td>Endosymbionts_Pairs03</td>
<td>19</td>
</tr>
<tr>
<td>Endosymbionts_Sets</td>
<td>19</td>
</tr>
<tr>
<td>Endosymbionts_Synteny</td>
<td>20</td>
</tr>
<tr>
<td>EstimateExoLabel</td>
<td>20</td>
</tr>
<tr>
<td>EstimRearrScen</td>
<td>22</td>
</tr>
<tr>
<td>EvoWeaver</td>
<td>25</td>
</tr>
<tr>
<td>EvoWeaver-GOPreds</td>
<td>28</td>
</tr>
<tr>
<td>EvoWeaver-PPPreds</td>
<td>29</td>
</tr>
<tr>
<td>EvoWeaver-PSPreds</td>
<td>31</td>
</tr>
<tr>
<td>EvoWeaver-SLPreds</td>
<td>33</td>
</tr>
<tr>
<td>EvoWeb</td>
<td>35</td>
</tr>
<tr>
<td>ExampleStreptomycesData</td>
<td>36</td>
</tr>
<tr>
<td>ExoLabel</td>
<td>36</td>
</tr>
<tr>
<td>ExpandDiagonal</td>
<td>40</td>
</tr>
<tr>
<td>ExtractBy</td>
<td>41</td>
</tr>
<tr>
<td>FastQFromSRR</td>
<td>43</td>
</tr>
<tr>
<td>FindSets</td>
<td>44</td>
</tr>
<tr>
<td>FitchParsimony</td>
<td>45</td>
</tr>
<tr>
<td>Generic</td>
<td>47</td>
</tr>
<tr>
<td>gffToDataFrame</td>
<td>48</td>
</tr>
<tr>
<td>LinkedPairs</td>
<td>49</td>
</tr>
<tr>
<td>MakeBlastDb</td>
<td>50</td>
</tr>
<tr>
<td>MoransI</td>
<td>51</td>
</tr>
<tr>
<td>NucleotideOverlap</td>
<td>53</td>
</tr>
<tr>
<td>PairSummaries</td>
<td>54</td>
</tr>
<tr>
<td>PhyloDistance</td>
<td>57</td>
</tr>
<tr>
<td>PhyloDistance-CIDist</td>
<td>59</td>
</tr>
<tr>
<td>PhyloDistance-JRFDist</td>
<td>60</td>
</tr>
<tr>
<td>PhyloDistance-KFDist</td>
<td>62</td>
</tr>
<tr>
<td>PhyloDistance-RFDist</td>
<td>63</td>
</tr>
<tr>
<td>plot.EvoWeb</td>
<td>64</td>
</tr>
<tr>
<td>predict.EvoWeaver</td>
<td>66</td>
</tr>
<tr>
<td>PrepareSeqs</td>
<td>69</td>
</tr>
</tbody>
</table>
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 either MakeBlastDb from R or 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 executed 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: Protein sequences against translated database of nucleotide sequences
• **blastx**: Translated nucleotide sequences against database of protein sequences
• **tblastx**: Translated nucleotide sequences against translated database of nucleotide 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 sequences for blastn, blastx, and tblastx should be nucleotide data.
Input sequences for blastp and tblastn should be amino acid data.
Database for blastn, tblastn, tblastx should be nucleotide data.
Database for blastp and blastx should be amino acid data.

**Value**

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

**Author(s)**

Aidan Lakshman <ahl27@pitt.edu>

**See Also**

MakeBlastDb

**Examples**

```
#
```

<table>
<thead>
<tr>
<th>BlockExpansion</th>
<th>Attempt to expand blocks of paired features in a PairSummaries object.</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
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**

```r
# this function will be deprecated soon,
# please see the new ExpandDiagonal() function.
library(RSQLite)
DBPATH <- system.file("extdata",
                   "Endosymbionts_v02.sqlite",
                   package = "SynExtend")
data("Endosymbionts_LinkedFeatures", package = "SynExtend")

Pairs <- PairSummaries(SyntenyLinks = Endosymbionts_LinkedFeatures,
                      PIDs = 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.
**BlockReconciliation**

**SCOREThreshold** 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 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 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 - frequencies 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**

```r
# this function will be deprecated soon...
## Not run:
data("Endosymbionts_Pairs02", package = "SynExtend")
Pairs03 <- BlockReconciliation(Pairs = Endosymbionts_Pairs02,
ConservativeRejection = FALSE,
Verbose = TRUE)

## End(Not run)
```

---

**BuiltInEnsembles**  
**Pretrained EvoWeaver Ensemble Models**

**Description**

EvoWeaver 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**

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

**Value**

The data contain a list of objects of class `glm`.

**Examples**

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

exData <- get(data("ExampleStreptomycesData"))
ew <- EvoWeaver(exData$Genes[1:50])
datavals <- predict(ew, 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 <- EvoWeaver(exData$Genes[51:60])
predict(predictionPW,
         PretrainedModel=myModel)
```
**Description**

Simulated values of Clustering Information Distance for random trees with 4 to 200 shared leaves.

**Usage**

```r
data("CIDist_NullDist")
```

**Details**

Each column of the matrix corresponds to the distribution of distances between random trees with the given number of leaves. This begins at `CI_DISTANCE_INTERNAL[,1]` corresponding to 4 leaves, and ends at `CI_DISTANCE_INTERNAL[,197]` corresponding to 200 leaves. Distances begin at 4 leaves since there is only one unrooted tree with 1, 2, or 3 leaves (so the distance between any given tree with less than 4 leaves is always 0).

Each row of the matrix corresponds to statistics for the given simulation set. The first row gives the minimum value, the next 9 give quantiles in `c(1%, 5%, 10%, 25%, 50%, 75%, 90%, 95%, 99%)`, and the last three rows give the `max`, `mean`, and `sd` (resp.).

**Value**

A matrix `CI_DISTANCE_INTERNAL` with 197 columns and 13 rows.

**Source**

Datafiles obtained from the TreeDistData package, published as part of Smith (2020).

**References**


**Examples**

```r
data(CIDist_NullDist)
```
ClusterByK

Predicted pair trimming using K-means.

Description
A relatively simple k-means clustering approach to drop predicted pairs that belong to clusters with a PID centroid below a specified user threshold.

Usage
ClusterByK(SynExtendObject,
UserConfidence = list("PID" = 0.3),
ClusterScalar = 4,
MaxClusters = 15L,
ColSelect = c("p1featurelength",
"p2featurelength",
"TotalMatch",
"Consensus",
"PID",
"Score"),
ColNorm = "Score",
ShowPlot = FALSE,
Verbose = FALSE)

Arguments
SynExtendObject
An object of class PairSummaries.

UserConfidence
A named list of length 1 where the name identifies a column of the PairSummaries object, and the value identifies a user confidence. Every k-means cluster with a center value of the column value selected greater than the confidence is retained.

ClusterScalar
A numeric value used to scale selection of how many clusters are used in kmeans clustering. A transformed total within-cluster sum of squares value is fit to a right hyperbola, and a scaled half-max value is used to select cluster number. “ClusterScalar” is multiplied by the half-max to adjust cluster number selection.

MaxClusters
Integer value indicating the largest number of clusters to test in a series of k-means clustering tests.

ColSelect
A character vector of column names indicating which columns to use for k-means clustering. When “p1featurelength”, “p2featurelength”, and “TotalMatch” are included together, they are morphed into a value representing the match size proportional to the longer of the two sequences.

ColNorm
A character vector of column names indicating columns the user would like to unit normalize. By default only set to “Score”.

ShowPlot
Logical indicating whether or not to plot the CDFs for the PIDs of all k-means clusters for the determined cluster number.
Verbose  Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

ClusterByK uses a naive k-means routine to select for predicted pairs that belong to clusters whose centroids are greater than or equal to the user specified column-value pair. This means that the confidence is not a minimum, and that pairs with values below the user confidence can be retained. The sum of within cluster sum of squares is used to approximate “knee” selection with the “ClusterScalar” value. With a “ClusterScalar” value of 1 the half-max of a right-hyperbola fitted to the sum of within-cluster sum of squares is used to pick the cluster number for evaluation, “ClusterScalar” is multiplied by the half-max to tune cluster number selection. ClusterByK returns the original object with an appended column and new attributes. The new column “ClusterID” is an integer value indicating which k-means cluster a candidate pair belongs to, while the attribute “Retain” is a named logical vector where the names correspond to ClusterIDs, and the logical value indicates whether the cluster center was above the user supplied column-value pair. This function is intended to be used at the genome-to-genome comparison level, and not say, at the level of an all-vs-all comparison of many genomes. It will work well in all-vs-all cases, but it is not optimized for that scale yet.

Value

An object of class PairSummaries.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

SummarizePairs, NucleotideOverlap, FindSynteny, ExpandDiagonal

Examples

data("Endosymbionts_Pairs01", package = "SynExtend")

Pairs02 <- ClusterByK(SynExtendObject = Endosymbionts_Pairs01)

dendrapply  Apply a Function to All Nodes of a Dendrogram

Description

Apply function FUN to each node of a dendrogram recursively. When y <- dendrapply(x, fn), then y is a dendrogram of the same graph structure as x and for each node, y.node[j] <- FUN( x.node[j], ...) (where y.node[j] is an (invalid!) notation for the j-th node of y). Also provides flexibility in the order in which nodes are evaluated.

NOTE: This man page is for the dendrapply function defined in the SynExtend package. See ?stats::dendrapply for the default method (defined in the stats package).
Usage

dendrapply(X, FUN, ..., how = c("pre.order", "post.order"))

Arguments

X          An object of class "dendrogram".
FUN        An R function to be applied to each dendrogram node, typically working on its attributes alone, returning an altered version of the same node.
...        potential further arguments passed to FUN.
how        one of c("pre.order", "post.order"), or an unambiguous abbreviation. Determines if nodes should be evaluated according to an preorder (default) or postorder traversal. See details for more information.

Details

"pre.order" preserves the functionality of the previous dendrapply. For each node n, FUN is applied first to n, then to n[[1]] (and any children it may have), then n[[2]] and its children, etc. Notably, each node is evaluated prior to any of its children.

"post.order" allows for calculations that depend on the children of a given node. For each node n, FUN is applied first to all children of n, then is applied to n itself. Notably, each node is evaluated after all of its children.

Value

Usually a dendrogram of the same (graph) structure as X. For that, the function must be conceptually of the form FUN <- function(X) { attributes(X) <- ....; X }, i.e., returning the node with some attributes added or changed.

If the function provided does not return the node, the result is a nested list of the same structure as X, or as close as can be achieved with the return values. If the function should only be applied to the leaves of X, consider using rapply instead.

Warning

dendrapply identifies leaf nodes as nodes such that attr(node, 'leaf') == TRUE, and internal nodes as nodes such that attr(node, 'leaf') %in% c(NULL, FALSE). If you modify or remove this attribute, dendrapply may perform unexpectedly.

Note

The prior implementation of dendrapply was recursive and inefficient for dendrograms with many non-leaves. This version is no longer recursive, and thus should no longer cause issues stemming from insufficient C stack size (as mentioned in the 'Warning' in dendrogram).

Author(s)

Aidan Lakshman <ahl27@pitt.edu>
See Also

as.dendrogram, lapply for applying a function to each component of a list.

rapply is particularly useful for applying a function to the leaves of a dendrogram, and almost always be used when the function does not need to be applied to interior nodes due to significantly better performance.

Examples

require(graphics)

## a smallish simple dendrogram
dhc <- as.dendrogram(hc <- hclust(dist(USArrests), "ave"))
(dhc21 <- dhc[[2]][[1]])

## too simple:
dendrapply(dhc21, function(n) utils::str(attributes(n)))

## toy example to set colored leaf labels :
local({
collab <- function(n) {
  if(is.leaf(n)) {
    a <- attributes(n)
    i <- i+1
    attr(n, "nodePar") <- c(a$nodePar, list(lab.col = mycols[i], lab.font = i%%3))
  }
  n
}
mycols <- grDevices::rainbow(attr(dhc21,"members"))
i <- 0
})
dL <- dendrapply(dhc21, collab)
op <- par(mfrow = 2:1)
plot(dhc21)
plot(dL) ## --> colored labels!
par(op)

## Illustrating difference between pre.order and post.order
dend <- as.dendrogram(hclust(dist(seq_len(4L))))

f <- function(x){
  if(is.null(attr(x, 'leaf'))){
    v <- as.character(attr(x, 'label'))
  } else {
    v <- paste0(attr(x[[1]], 'newattr'), attr(x[[2]], 'newattr'))
  }
  attr(x, 'newattr') <- v
  x
}

# trying with default, note character(0) entries
preorder_try <- dendrapply(dend, f)
dendrapply(preorder_try, \(x\){ print(attr(x, 'newattr')); x })

## trying with postorder, note that children nodes will already
## have been populated, so no character(0) entries
postorder_try <- dendrapply(dend, f, how='post.order')
dendrapply(postorder_try, \(x\){ print(attr(x, 'newattr')); x })

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)
**Description**

Calculates if a presence/absence pattern is random, Brownian, or neither with respect to a given phylogeny.

**Usage**

DPhyloStatistic(dend, PAProfile, NumIter = 1000L)

**Arguments**

- **dend**: An object of class `dendrogram`
- **PAProfile**: A vector representing presence/absence of binary traits. See Details for more information.
- **NumIter**: Number of iterations to simulate for random permutation analysis.

**Details**

This function implements the D-Statistic for binary traits on a phylogeny, as introduced in Fritz and Purvis (2009). The statistic is the following ratio:

\[
\frac{D_{\text{obs}} - D_b}{D_r - D_b}
\]

Here \(D_{\text{obs}}\) is the D value for the input data, \(D_b\) is the value under simulated Brownian evolution, and \(D_r\) is the value under random permutation of the input data. The D value measures the sum of sister clade differences in a phylogeny weighted by branch lengths. A score close to 1 indicates phylogenetically random distribution, and a score close to 0 indicates the trait likely evolved under Brownian motion. Scores can fall outside this range; these scores are only intended as benchmark points on the scale. See the original paper cited in References for more information.

The input `PAProfile` supports a number of formatting options:

- Character vector, where each element is a label of the dendrogram. Presence in the character vector indicates presence of the trait in the corresponding label.
- Integer vector of length equivalent to the number of leaves, comprised of 0s and 1s. 0 indicates absence in the corresponding leaf, and 1 indicates presence.
- Logical vector of length equivalent to number of leaves. `FALSE` indicates absence in the corresponding leaf, and `TRUE` indicates presence.

See Examples for a demonstration of each case.

**Value**

Returns a numerical value. Values close to 0 indicate random distribution, and values close to 1 indicate a Brownian distribution.
**Author(s)**
Aidan Lakshman <ahl27@pitt.edu>

**References**

**Examples**

```r
#########################################################
### Replicating results from Table 1 in original paper ###
#########################################################

# creates a dendrogram with 16 leaves and branch lengths all 1
distMat <- suppressWarnings(matrix(1:17, nrow=16, ncol=16))
testDend <- as.dendrogram(hclust(as.dist(distMat)))
testDend <- dendrapply(testDend, function(x){
  attr(x, 'height') <- attr(x, 'height') / 2
  return(x)
})
attr(testDend[[1]], 'height') <- attr(testDend[[2]], 'height') <- 3
attr(testDend, 'height') <- 4
plot(testDend)

set.seed(123)
# extremely clumped (should be close to -2.4)
DPhyloStatistic(testDend, as.character(1:8))

# clumped Brownian (should be close to 0)
DPhyloStatistic(testDend, as.character(c(1,2,5,6,10,12,13,14)))

# random (should be close to 1.0)
DPhyloStatistic(testDend, as.character(c(1,4,6,10,13,14,16)))

# overdispersed (should be close to 1.9)
DPhyloStatistic(testDend, as.character(seq(2,16,by=2)))

#########################################################
### Different ways to create PAProfiles ###
#########################################################

allLabs <- as.character(labels(testDend))

# All these ways create a PAProfile with
# presence in members 1:4
# and absence in members 5:16
# numeric vector:
c(rep(1,4), rep(0, length(allLabs)-4))
```
# logical vector:
c(rep(TRUE,4), rep(FALSE, length(allLabs)-4))

# character vector:
allLabs[1:4]
Value

An object of class LinkedPairs.

Examples

data(Endosymbionts_LinkedFeatures)


Endosymbionts_Pairs01  Example predicted pairs

Description

An object of class PairSummaries.

Usage

data("Endosymbionts_Pairs01")

Details

An object of class PairSummaries.

Value

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")

Details

An object of class PairSummaries.
Endosymbionts_Pairs03

Value
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")

Details
An object of class PairSummaries.

Value
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")

Details
A named list of disjoint sets representing hypothetical COGs.
EstimateExoLabel

Value

A named list of disjoint sets representing hypothetical COGs.

Examples

```r
data(Endosymbionts_Sets)
```

---

Endosymbionts_Synteny  A synteny object

**Description**

An object of class Synteny.

**Usage**

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

**Details**

An object of class Synteny.

**Value**

An object of class Synteny.

**Examples**

```r
data(Endosymbionts_Synteny)
```

---

EstimateExoLabel  Estimate ExoLabel Disk Consumption

**Description**

Estimate the total disk consumption for ExoLabel.

**Usage**

```r
EstimateExoLabel(num_v, avg_degree=1,
num_edges=num_v*avg_degree,
node_name_length=8L)
```
EstimateExoLabel

Arguments
num_v                  Approximate number of total unique nodes in the network.
avg_degree             Average degree of each node in the network.
num_edges              Approximate total number of edges in the network.
node_name_length       Approximate average length of each node name, in characters.

Details
This function provides a rough estimate of the total disk space required to run ExoLabel for a given input network. `avg_degree` and `num_edges` need not both be specified. The function prints out the estimated size of the original edgelist files, the estimated disk space to be consumed by ExoLabel, and the approximate ratio of disk space relative to the original file.

`node_name_length` specifies the average length of the node names—since the names themselves must be stored on disk, this contributes to the overall size. For relatively short node names (1-16 characters) this has a negligible impact on overall disk consumption.

Value
Returns a vector of length three, showing the estimated total edgelist file size, estimated disk consumption, and ratio of the two. All sizes are shown in bytes.

Note
Estimating the average node label size is challenging, and unfortunately it does have a relatively large effect on the estimated edgelist file size. This function should be used for rough estimations of sizing, not absolute values. Errors in estimation of rough node name size will have a larger impact on edgelist file estimation than on the ExoLabel disk usage, so users can have higher confidence in estimated ExoLabel consumption.

Author(s)
Aidan Lakshman <AHL27@pitt.edu>

See Also
ExoLabel

Examples
# 100,000 nodes, average degree 2
EstimateExoLabel(num_v=100000, avg_degree=2)

# 10,000 nodes, 50,000 edges
EstimateExoLabel(num_v=10000, num_edges=50000)
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  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SyntenyObject</td>
<td>Synteny object, as obtained from running FindSynteny. Expected input is unichromosomal sequences, though multichromosomal sequences are supported.</td>
</tr>
<tr>
<td>NumRuns</td>
<td>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.</td>
</tr>
<tr>
<td>Mean</td>
<td>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.</td>
</tr>
<tr>
<td>MinBlockLength</td>
<td>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.</td>
</tr>
<tr>
<td>Verbose</td>
<td>Logical; indicates whether or not to display a progress bar and print the time difference upon completion.</td>
</tr>
</tbody>
</table>

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 occurring half as often as gene inversions.

This implementation incentivizes the 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 \(N \times N\) matrix of lists with the same shape as the input Syteny object. This is wrapped into a \texttt{GenRearr} object for pretty printing.

The diagonal corresponds to the 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 \texttt{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 sequence 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 \(1,2,\ldots,N\), and then have genome 2 numbers relative to that.

As an example, suppose genome 1 has elements \(ABE(r)G\) and genome 2 has elements \(EB(r)\ A(r)\ G\), 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 $\text{Scenario}$ 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 $\text{Scenario}$ 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 $\text{Scenario}$ 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 $\text{Scenario}$ 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.

$\text{Key}$ 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 $\text{Scenario}$ output

**Author(s)**

Aidan Lakshman (<ahl27@pitt.edu>)

**References**


**See Also**

- FindSynteny
- Synteny

**Examples**

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

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

---

**EvoWeaver**

**EvoWeaver: Predicting Protein Functional Association Networks**

**Description**

EvoWeaver is an S3 class with methods for predicting functional association using protein or gene data. EvoWeaver implements multiple algorithms for analyzing coevolutionary signal between genes, which are combined into overall predictions on functional association. For details on predictions, see `predict.EvoWeaver`.

**Usage**

EvoWeaver(ListOfData, MySpeciesTree=NULL, NoWarn=FALSE)

## S3 method for class 'EvoWeaver'
SpeciesTree(ew, Verbose=TRUE, Processors=1L)

**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.

- **MySpeciesTree**
  An object of class 'dendrogram' representing the overall species tree for the list provided in ListOfData.

- **NoWarn**
  Several algorithms depend on having certain data. When a EvoWeaver object is initialized, it automatically selects which algorithms can be used given the input data. By default, EvoWeaver will notify the user of algorithms that cannot be used with warnings. Setting `NoWarn=TRUE` will suppress these messages.

- **ew**
  An object of class EvoWeaver

- **Verbose**
  Should output be displayed when calculating species tree?

- **Processors**
  Number of processors to use. Set to NULL to automatically use the maximum amount of processors.

**Details**

EvoWeaver 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. (a) \texttt{ListOfData[[i]] = c('i1_d1_p1', 'i2_d2_p2', ..., 'ik_dk_pk')}
   (b) \texttt{ListOfData[[i]] = c('i1_d1_s1_p1', 'i2_d2_s2_p2', ..., 'ik_dk_sk_pk')}
3. \texttt{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 (2a) is the same as (1), just with the formatting of names slightly different. Each entry is of the form \texttt{i_d_p}, where \texttt{i} is the unique identifier for the genome, \texttt{d} is which chromosome the ortholog is located, and \texttt{p} is what position the ortholog appears in on that chromosome. \texttt{p} must be a numeric, while the other entries can be any value.

Case (2b) is a variation on (2a), adding in an identifier \texttt{s}. This value must be 0 or 1, corresponding to whether the gene is on the forward or reverse strand. Whether 0 denotes forward or reverse is inconsequential as long as the scheme is consistent.

Case (3) expects gene trees for each gene, with labeled leaves corresponding to each source genome. If \texttt{ListOfData} is in this format, taking \texttt{labels(ListOfData[[i]])} should produce a character vector that matches the format of one of the previous cases.

See the Examples section for illustrative examples.

Whenever possible, provide a full set of dendrogram objects with leaf labels in form (2b). This will allow the most algorithms to run. What follows is a more detailed description of which inputs allow which algorithms.

EvoWeaver requires input of scenario (3) to use distance matrix methods, and requires input of scenario (2) (or (3) with leaves labeled according to (2)) for gene organization analyses. Transcriptional direction analysis requires input of scenario (2b). Sequence-level methods require dendrograms with sequence information included as the state attribute in each leaf node.

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 \texttt{predict(EvoWeaverObject)}. See \texttt{predict.EvoWeaver} for more information.

The \texttt{SpeciesTree} function takes in an object of class \texttt{EvoWeaver} and returns a species tree. If the object was not initialized with a species tree, it calculates one using \texttt{SuperTree}. The species tree for a \texttt{EvoWeaver} object can be set with \texttt{attr(ew, 'speciesTree')} <- ... .

\textbf{Value}

Returns a \texttt{EvoWeaver} object.

\textbf{Author(s)}

Aidan Lakshman <ahl27@pitt.edu>

\textbf{See Also}

\texttt{predict.EvoWeaver, ExampleStreptomycesData, BuiltInEnsembles, SuperTree}
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 EvoWeaver 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 <- EvoWeaver(l)

## Constructing a EvoWeaver 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_0', '3_1_2', '4_1_1')
l[['b']] <- c('1_1_1', '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')
pwCase2a <- EvoWeaver(l)

## If we want transcriptional information, we need an
## value corresponding to the strand of each gene
## Notice that the genome identifier need not be numeric,
## but the strand identifier must be 0 or 1
1 <- list()
l[['a']] <- c('a_1_0_1', 'c_1_1_2', 'd_1_0_1')
EvoWeaver-GOPreds

Gene Organization Predictions for EvoWeaver

Description

EvoWeaver incorporates four classes of prediction, each with multiple methods and algorithms. Co-localization (Coloc) methods examine conservation of relative location and transcriptional direction of genetic regions within the genome.

predict.EvoWeaver currently supports three Coloc methods:

- 'Coloc'
- 'ColocMoran'
- 'TranscripMI'

Details

All distance matrix methods require a EvoWeaver object initialized with gene locations using the a three or four number code. See EvoWeaver for more information on input data types.

The built-in Coloc examines relative location of genes within genomes as evidence of interaction. For a given pair of genes, the score is given by \( \sum_{G} e^{dI_G} \), where \( G \) the set of genomes and \( dI_G \) the difference in index between the two genes in genome \( G \). Using gene index instead of number of base pairs avoids bias introduced by gene and genome length.

ColocMoran measures the extent to which gene distances are preserved across a phylogeny. This function uses the same initial scoring scheme as Coloc, but can handle paralogs. The raw scores are passed into MoransI to calculate spatial autocorrelation. "Space" is taken as \( e^{-C} \), where \( C \) is the Cophenetic distance matrix calculated from the species tree of the inputs. As such, this method requires a species tree as input, which can be calculated from a set of gene trees using SuperTree.

TranscripMI uses mutual information of the transcriptional direction of each pair of genes. Conservation of relative transcriptional direction between gene pairs has been shown to imply functional association in prior work. This algorithm requires that the EvoWeaver object is initialized with a four number code, with the third number either 0 or 1, denoting whether the gene is on the forward or reverse strand. The mutual information is calculated as:
\[
\sum_{x \in X} \sum_{y \in Y} (-1)^{(x! = y)} P_{X,Y}(x, y) \log \left( \frac{P_{X,Y}(x, y)}{P_X(x)P_Y(y)} \right)
\]

Here \( X = Y = \{0, 1\} \), \( x \) is the direction of the gene with lower index, \( y \) is the direction of the gene with higher index, and \( P_{T}(t) \) is the probability of \( T = t \). Note that this is a weighted MI as introduced by Beckley and Wright (2021). The mutual information is augmented by the addition of a single pseudocount to each value, and normalized by the joint entropy of \( X, Y \). P-values are calculated using Fisher’s Exact Test on the contingency table.

**Value**

None.

**Author(s)**

Aidan Lakshman <ahl27@pitt.edu>

**References**


**See Also**

EvoWeaver

predict.EvoWeaver

EvoWeaver Phylogenetic Profiling Predictors

EvoWeaver Phylogenetic Structure Predictors

EvoWeaver Sequence-Level Predictors

---

**Description**

EvoWeaver incorporates four classes of prediction, each with multiple methods and algorithms. Phylogenetic Profiling (PP) methods examine conservation of gain/loss events within orthology groups using phylogenetic profiles constructed from presence/absence patterns.

predict.EvoWeaver currently supports seven PP methods:

- 'Jaccard'
• 'Hamming'
• 'MutualInformation'
• 'PAPV'
• 'CorrGL'
• 'ProfDCA'
• 'Behdenna'
• 'GainLoss'

Details

Most PP methods are compatible with a EvoWeaver object initialized with any input type. See EvoWeaver for more information on input data types.

All of these methods use presence/absence (PA) profiles, which are binary vectors such that 1 implies the corresponding genome has that particular gene, and 0 implies the genome does not have that particular gene.

Methods Hamming and Jaccard use Hamming and Jaccard distance (respectively) of PA profiles to determine overall score.

MutualInformation uses mutual information of PA profiles to determine score, employing a weighting scheme such that 11 and 00 give positive information, and 10 and 01 give negative information.

PAPV calculates a p-value for PA profiles using Fisher’s Exact Test. The returned score is provided as 1-p_value so that larger scores indicate more significance, and smaller scores indicate less significance. This rescaling is consistent with the other similarity metrics in EvoWeaver. This can be used with Jaccard, Hamming, or MutualInformation to weight raw scores by statistical significance.

ProfDCA uses the direct coupling analysis algorithm introduced by Weigt et al. (2005) to determine direct information between PA profiles. This approach has been validated on PA profiles in Fukunaga and Iwasaki (2022), though the implementation in EvoWeaver forsakes the persistent contrastive divergence method in favor of the the algorithm from Lokhov et al. (2018) for increased speed and exact solutions. Note that this algorithm is still extremely slow relative to the other methods despite the aforementioned runtime improvements.

Behdenna implements the method detailed in Behdenna et al. (2016) to find statistically significant interactions using co-occurrence of gain/loss events mapped to ancestral states on a species tree. This method requires a species tree as input. If the EvoWeaver object is initialized with dendrogram objects, SuperTree will be used to infer a species tree.

GainLoss uses a similar method to Behdenna. This method uses Fitch Parsimony to infer where events were gained or lost on a species tree, and then looks for distance between these gain/loss events. Unlike Behdenna, this method takes into account the types of events (ex. gain/gain and loss/loss are treated differently than gain/loss). This method requires a species tree as input. If the EvoWeaver object is initialized with dendrogram objects, SuperTree will be used to infer a species tree.

CorrGL infers where events were gained or lost on a species tree as in method GainLoss, then uses a Pearson’s correlation coefficient weighted by p-value to infer similarity.

Value

None.
EvoWeaver incorporates four classes of prediction, each with multiple methods and algorithms. Phylogenetic Structure (PS) methods examine conservation of overall evolutionary rates within orthology groups using distance matrices constructed from each gene tree.

predict.EvoWeaver currently supports three PS methods:

- 'MirrorTree'
- 'ContextTree'
- 'TreeDistance'
Details

All distance matrix methods require a EvoWeaver object initialized with dendrogram objects. See EvoWeaver for more information on input data types.

The MirrorTree method was introduced by Pazos et al. (2001). This method builds distance matrices using a nucleotide substitution model, and then calculates coevolution between gene families using the Pearson correlation coefficient of the upper triangle of the two corresponding matrices. Experimental analysis has shown data in the upper triangle is heavily redundant and rapidly overwhelms available system memory. Previous work has incorporated dimensionality reduction such as SVD to reduce the dimensionality of the data, but this prevents parallelization of the data and doesn’t solve memory issues (since SVD takes as input the entire matrix with columns corresponding to upper triangle values). EvoWeaver instead uses a seeded random projection following Achlioptas (2001) to reduce the dimensionality of the data in a reproducible and parallel-compatible way. We also utilize Spearman’s \( \rho \), which outperforms Pearson’s \( r \) following dimensionality reduction.

Subsequent work by Pazos et al. (2005) and Sato et al. (2005, 2006) found multiple ways to improve predictions from the initial MirrorTree method. These methods incorporate additional phylogenetic context, and are thus called ContextTree methods. These improvements include correcting for overall evolutionary rate using a species tree and/or using projection vectors. The built-in ContextTree method implements a species tree correction, and weights the resulting score by the normalized Hamming distance of the presence/absence profiles. This can correct for gene trees with low overlap that achieve spuriously high scores via random projection. Additional correction measures are implemented in the MTCorrection argument.

The TreeDistance method uses phylogenetic tree distance to quantify differences between gene trees. This method implements a number of metrics and groups them together to improve overall runtime. The default tree distance method is normalized Robinson-Foulds distance due to its lower computational complexity. Other methods can be specified using the TreeMethods argument, which expects a character vector containing one or more of the following:

- "CI": Clustering Information Distance
- "RF": Robinson-Foulds Distance
- "JRF": Jaccard-Robinson-Foulds Distance
- "Nye": Nye Similarity
- "KF": Kuhner-Felsenstein Distance
- "all": All of the above methods

See the links above for more information and references. All of these metrics are accessible using the PhyloDistance method. Method "JRF" defaults to a k value of 4, but this can be specified further if necessary using the JRFk input parameter. Higher values of k approach the value of Robinson-Foulds distance, but these have a negligible impact on performance so use of the default parameter is encouraged for simplicity. Multiple metrics can be specified.

Value

None.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>
References


See Also

- EvoWeaver
- predict.EvoWeaver
- EvoWeaver Phylogenetic Profiling Predictors
- EvoWeaver Gene Organization Predictors
- EvoWeaver Sequence-Level Predictors
- PhyloDistance

Description

EvoWeaver incorporates four classes of prediction, each with multiple methods and algorithms.

Sequence-Level (SL) methods examine conservation of patterns in sequence data, commonly exhibited due to physical interactions between proteins.

predict.EvoWeaver currently supports three SL methods:

- 'ResidueMI'
- 'NVDT'
- 'Ancestral'

Details

All residue methods require an EvoWeaver object initialized with dendrogram objects and ancestral states. See EvoWeaver for more information on input data types.

The ResidueMI method looks at mutual information between sites in a multiple sequence alignment (MSA). This approach extends prior work in Martin et al. (2005). Each site from the first gene group is paired with the site from the second gene group that maximizes their mutual information.
The NVDT method uses the natural vector encoding method introduced in Zhao et al. (2022). This encodes each gene sequences as a 92-dimensional vector, with the following entries:

\[ N(S) = (n_A, n_C, n_G, n_T, \mu_A, \mu_C, \mu_G, \mu_T, D_A^2, D_C^2, D_G^2, D_T^2, n_{AA}, n_{AC}, \ldots, n_{TT}) \]

Here \( n_X \) is the raw total count of nucleotide \( X \) (or di/trinucleotide). For single nucleotides, we also calculate \( \mu_X \), the mean location of nucleotide \( X \), and \( D_X^2 \), the second moment of the location of nucleotide \( X \). The overall natural vector for a COG is calculated as the normalized mean vector from the natural vectors of all component gene sequences. Interaction scores are computed using Pearson’s R between each COG’s natural vector. These di/trinucleotide counts are by default excluded, but can be included using the extended=TRUE argument. Using the extended counts has shown minimal increased accuracy at the cost of slower runtime in benchmarking.

The Ancestral method calculates coevolution by looking at correlation of residue mutations near the leaves of each respective gene tree.

Value

None.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

References


See Also

EvoWeaver
predict.EvoWeaver
EvoWeaver Phylogenetic Profiling Predictors
EvoWeaver Phylogenetic Structure Predictors
EvoWeaver Gene Organization Predictors
Description

EvoWeb objects are outputted from `predict.EvoWeaver`. This class wraps the `simMat` object with some other diagnostic information intended to help interpret the output of `EvoWeaver` predictions.

Details

`predict.EvoWeaver` returns a EvoWeb object, which bundles some methods to make formatting and printing of results slightly nicer. This currently only implements a `plot` function, but future functionality is in the works.

Value

An object of class "EvoWeb", which inherits from "simMat".

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

See Also

`predict.EvoWeaver`
`simMat`
`plot.EvoWeb`

Examples

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

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

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

evoweb <- predict(ew, Method='Jaccard')

# print out results as an adjacency matrix
print(evoweb)

# print out results as a pairwise data.frame
as.data.frame(evoweb)
```
ExampleStreptomycesData

*Example EvoWeaver Input Data from Streptomyces Species*

**Description**

Data from Streptomyces species to test *EvoWeaver* functionality.

**Usage**

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

**Details**

This dataset contains a number of Clusters of Orthologous Genes (COGs) and a species tree for use with *EvoWeaver*. This dataset showcases an example of using *EvoWeaver* 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.

**Value**

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

**See Also**

*EvoWeaver*

**Examples**

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

---

ExoLabel

*ExoLabel: Out of Memory Fast Label Propagation*

**Description**

Runs Fast Label Propagation using disk space for constant memory complexity.
Usage

ExoLabel(edgelistfiles, outfile=tempfile(),
mode=c("undirected", "directed"),
add_self_loops=FALSE,
ignore_weights=FALSE,
normalize_weights=FALSE,
itereations=0L,
return_table=FALSE,
consensus_cluster=FALSE,
verbose=interactive(),
sep='t',
tempfiledir=tempdir(),
cleanup_files=TRUE)

Arguments

edgelistfiles  Character vector of files to be processed. Each entry should be a machine-
interpretable path to an edgelist file. See Details for expected format.

outfile  File to write final clusters to. Optional, defaults to a temporary file.

mode  String specifying whether edges should be interpreted as undirected (default) or
directed. Can be "undirected", "directed", or an unambiguous abbreviation.

add_self_loops  Should self loops be added to the network? If TRUE, adds self loops of weight 1.0
to all vertices. If set to numeric value w, adds self loops of weight w to all nodes.
Note that this takes place prior to edge weight normalization (if requested).

ignore_weights  Should weights be ignored? If TRUE, all edges will be treated as an edge of
weight 1. Must be set to TRUE if any of edgelistfiles are two-column tables
(start->end only, lacking a weights column).

normalize_weights  Should weights be normalized? If TRUE, each vertex's edge weights are normalized
such that the outgoing edges have weight 1. This normalization is done after adding self loops.

iterations  Number of iterations to run fast label propagation algorithm for. Set to a value
of 0 or less for infinite iterations.

return_table  Should result of clustering be returned as a file, or a data.frame object? If
FALSE, returns a character vector corresponding to the path of outfile. If TRUE,
parses outfile using read.table and returns the result. Not recommended for
very large graphs.

consensus_cluster  Should consensus clustering be used? If TRUE, runs the clustering algorithm
nine times and forms a consensus clustering based on the agreement of each
run. Can be set to a double to control the number of iterations, see Details for
more information.

verbose  Should status messages (output, progress, etc.) be displayed while running?

sep  Character that separates entries on a line in each file in edgelistfiles. De-
defaults to tab, as would be expected in a .tsv formatted file. Set to ',' for a
.csv file.
tempfiledir Character vector corresponding to the location where temporary files used during execution should be stored. Defaults to R’s tempdir.

cleanup_files Should intermediary files be deleted when the process completes? Note that outfile will only be deleted if return_table=TRUE AND cleanup_files=TRUE.

Details

Very large graphs require too much RAM for processing on some machines. In a graph containing billions of nodes and edges, loading the entire structure into RAM is rarely feasible. This implementation uses disk space for storing representations of each graph. While this is slower than computing on RAM, it allows this algorithm to scale to graphs of enormous size while only taking a small amount of RAM. Local benchmarking resulting in a plateau of approximately 100MB of RAM consumption for arbitrarily sized networks. If your graph is small enough to fit into RAM, consider using the LP_igraph() function instead.

This function expects a set of edgelist files, provided as a vector of filepaths. Each entry in the file is expected to be in the following:

VERTEX1<sep>VERTEX2<sep>WEIGHT<linesep>

This line defines a single edge between vertices VERTEX1 and VERTEX2 with weight WEIGHT. VERTEX1 and VERTEX2 are strings corresponding to vertex names, WEIGHT is a numeric value that can be interpreted as a double. The separators <sep> and <linesep> correspond to the arguments sep and linesep, respectively. The default arguments work for standard .tsv formatting, i.e., a file of three columns of tab-separated values.

If ignore_weight=TRUE, the file can be formatted as:

VERTEX1<sep>VERTEX2<linesep>

Note that the v1 v2 w format is still accepted for ignore_weight=FALSE, but the specified weights will be ignored.

Consensus clustering can be enabled by setting consensus_cluster=TRUE. Consensus clustering runs this algorithm on the graph multiple times, transforming weight values according to a sigmoid function. By default, this runs nine times for sigmoids with scale 0.5 and shape c(0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.33, 1.67, 2.0), collapsing weights below 0.1 to zero. The resulting clusters form a network such that the edge weight between any two nodes connected in the initial graph is the proportion of clusters they shared over clustering runs. This network is used for a final label propagation run, which identifies the consensus clusters. Users can specify a numeric vector as input to consensus_cluster, which will override the default shape parameters and number of iterations.

Value

If return_table=TRUE, returns a data.frame object with two columns. The first column contains the name of each vertex, and the second column contains the cluster it was assigned to.

If return_table=FALSE, returns a character vector of length 1. This vector contains the path to the file where clusters were written to. The file is formatted as a .tsv, with each line containing two tab separated columns (vertex name, assigned cluster)

Warning

While this algorithm can scale to very large graphs, it does have some internal limitations. First, nodes must be comprised of no more than 254 characters. If this limitation is restrictive, please feel
free to contact me. Alternatively, you can increase the size yourself by changing the definition of MAX_NODE_NAME_SIZE in src/outmem_graph.c. This limitation is provided to decrease memory overhead and improve runtime, but arbitrary values are possible.

Second, nodes are indexed using 64-bit unsigned integers, with 0 reserved for other values. This means that the maximum possible number of nodes available is $2^{64}-2$, which is about 18.5 quintillion.

Third, this algorithm uses disk space to store large objects. As such, please ensure you have sufficient disk space for the graph you intend to process. I’ve tried to put safeguards in the code itself, but funky stuff can happen when the OS runs out of space.

The disk space required by this algorithm is equivalent to $56V + LV + 16E$, where $V$ the number of unique vertices, $L$ the average length of each vertex label, and $E$ the number of edges. We can instead represent this purely in terms of vertices by setting $D$ to the average degree of each node. The total disk space required is thus $(56 + L + 16D)V$. If you’d like to estimate the disk consumption of this algorithm, use the `EstimateExoLabel` function.

**Author(s)**

Aidan Lakshman <AHL27@pitt.edu>

**References**


**See Also**

`EstimateExoLabel`

**Examples**

```r
num_verts <- 20L
num_edges <- 20L
all_verts <- sample(letters, num_verts)
all_edges <- vapply(seq_len(num_edges),
  \(i\) paste(c(sample(all_verts, 2L),
             as.character(round(runif(1),3))),
             collapse='\t'),
  character(1L))
edgefile <- tempfile()
if(file.exists(edgefile)) file.remove(edgefile)
writeLines(all_edges, edgefile)
res <- ExoLabel(edgefile, return_table=TRUE)
print(res)
```
ExpandDiagonal  

**Description**

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

**Usage**

```r
ExpandDiagonal(SynExtendObject, FeatureSeqs, DataBase, InheritConfidence = TRUE, GapTolerance = 100L, DropSingletons = FALSE, UserConfidence = list("PID" = 0.3), Verbose = FALSE)
```

**Arguments**

- `SynExtendObject`: An object of class PairSummaries.
- `FeatureSeqs`: An object of class FeatureSeqs.
- `DataBase`: A character string pointing to a SQLite database, or a connection to a DECIPHER database.
- `InheritConfidence`: A logical indicating whether or not to inherit the user specified column-value pairs assigned to the input object.
- `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 100L.
- `DropSingletons`: Ignore solo pairs when planning expansion routes. Set to FALSE by default.
- `UserConfidence`: A named list of length 1 where the name identifies a column of the PairSummaries object, and the value identifies a user confidence. Every k-means cluster with a center value of the column value selected greater than the confidence is retained.
- `Verbose`: Logical indicating whether or not to display a progress bar and print the time difference upon completion.

**Details**

ExpandDiagonal 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.
ExtractBy

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

PairSummaries, NucleotideOverlap, link{SubSetPairs}, FindSynteny

Examples

```r
DBPATH <- system.file("extdata", "Endosymbionts_v02.sqlite", package = "SynExtend")
data("Endosymbionts_LinkedFeatures", package = "SynExtend")
Endosymbiont_Seqs <- PrepareSeqs(SynExtendObject = Endosymbionts_LinkedFeatures,
                                 DataBase = DBPATH,
                                 Verbose = TRUE)
data("Endosymbionts_Pairs02", package = "SynExtend")
Pairs03 <- ExpandDiagonal(SynExtendObject = Endosymbionts_Pairs02,
                          DataBase = DBPATH,
                          FeatureSeqs = Endosymbiont_Seqs,
                          Verbose = TRUE)
```

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 DNAStringSet 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

```r
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_v02.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)
FastQFromSRR  Get Sequencing Data from the SRA

Description
Get sequencing data from the SRA.

Usage
FastQFromSRR(SRR, 
ARGS = list("--gzip" = NULL, 
"--skip-technical" = NULL, 
"--readids" = NULL, 
"--read-filter" = "pass", 
"--dumpbase" = NULL, 
"--split-3" = NULL, 
"--clip" = NULL), 
KEEPFILES = FALSE)

Arguments
SRR  A character vector of length 1 representing an SRA Run Accession, such as one that would be passed to the prefetch, fastq-dump, or fasterq-dump functions in the SRAToolkit.

ARGS  A list representing key and value sets used to construct the call to fastq-dump, multi-argument values are passed to paste directly and should be structured accordingly.

KEEPFILES  Logical indicating whether or not keep the downloaded fastq files outside of the R session. If TRUE, downloaded files will be moved to R’s working directory with the default names assigned by fastq-dump. If FALSE - the default, they are removed and only the list of QualityScaledDNAStringSets returned by the function are retained.

Details
FastQFromSRR is a barebones wrapper for fastq-dump, it is set up for convenience purposes only and does not add any additional functionality. Requires a functioning installation of the SRAToolkit.

Value
A list of QualityScaledDNAStringSets. The composition of this list will be determined by fastq-dump’s splitting arguments.

Author(s)
Nicholas Cooley <npc19@pitt.edu>
Examples

```r
x <- "ERR10466327"
y <- FastQFromSRR(SRR = x)
```

---

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

```r
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

```r
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])
```

---

**FitchParsimony**

*Calculate ancestral states using Fitch Parsimony*

Description

Ancestral states for binary traits can be inferred from presence/absence patterns at the tips of a dendrogram using Fitch Parsimony. This function works for an arbitrary number of states on bifurcating dendrogram objects.

Usage

```r
FitchParsimony(dend, num_traits, traits_list,
               initial_state=rep(0L,num_traits),
               fill_ambiguous=TRUE)
```

Arguments

dend An object of class 'dendrogram'

num_traits The number of traits to inferred, as an integer.

traits_list A list of character vectors, where the i’th entry corresponds to the leaf labels that have the trait i.

initial_state The state assumed for the root node. Set to NULL to disable autofilling the root state.

fill_ambiguous If TRUE, states that remain ambiguous after completion of the algorithm are filled in randomly.
Details

Fitch Parsimony allows for fast inference of ancestral states of binary traits. The algorithm proceeds in three steps.

First, traits are inferred upwards based on child nodes. If the child nodes have the same state (1/1 or 0/0), then the parent node is also set to that state. If the states are different, the parent node is set to 2, denoting an ambiguous entry. If one child is ambiguous and the other is not, the parent is set to the non-ambiguous entry.

Second, traits are inferred downward to attempt to fill in ambiguous entries. If a node is not ambiguous but its child is, the child’s state is set to the parent state. If specified, the root node’s state is set to initial_state prior to this step.

Third, traits that remain ambiguous are optionally filled in (only if fill_ambiguous is set to TRUE). This proceeds by randomly setting ambiguous traits to either 1 or 0.

The result is stored in the FitchState attribute within each node.

Value

A dendrogram with attribute FitchState set for each node, where this attribute is a binary vector of length num_traits.

Note

It’s FitchParsimony because this implementation is entirely in R, as opposed to internal SynExtend methods that utilize a slightly faster C-based implementation that is not user-exposed.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

References


Examples

d <- as.dendrogram(hclust(dist(USArrests), "ave"))
labs <- labels(d)

    # Defining some presence absence patterns
    set.seed(123L)
pa_1 <- sample(labs, 15L)
pa_2 <- sample(labs, 20L)

    # inferring ancestral states
    fpd <- FitchParsimony(d, 2L, list(pa_1, pa_2))

    # Checking a state
    attr(fpd[[1L]], 'FitchState')
# Visualizing the results for the first pattern
# Tips show P/A patterns, edges show gain/loss (green/red)
fpd <- dendrapply(fpd, \(x\){
  ai <- 1L
  s <- attr(x, 'FitchState')
  l <- list()
  if(is.leaf(x)){
    # coloring tips based presence/absence
    l$col <- ifelse(s[ai]==1L, 'green', 'red')
    l$pch <- 19
    attr(x, 'nodePar') <- l
  } else {
    # coloring edges based on gain/loss
    for(i in seq_along(x)){
      sc <- attr(x[[i]], 'FitchState')
      if(s[ai] != sc[ai]){<br>
        l$col <- ifelse(s[ai] == 1L, 'red', 'green')
      } else {
        l$col <- 'black'
      }
      attr(x[[i]], 'edgePar') <- l
    }
  }
} }, how='post.order')
plot(fpd, leaflab='none')

---

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

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

#### Details

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

#### Value

The format is an object of class “glm”. 
gffToDataFrame

Examples

```r
data(Generic)
```

#### Description

Generate a DataFrame of gene calls from a gff3 file

#### Usage

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

#### Arguments

- **GFF**
  A url or filepath specifying a gff3 file to import

- **AdditionalAttrs**

- **AdditionalTypes**
  A vector of character strings to query from 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 “tRNA”, 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,
    ...)
MakeBlastDb

Create a BLAST Database from R

Description

Wrapper to create BLAST databases for subsequent 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

MakeBlastDb(seqs, dbtype=c('prot', 'nucl'),
            dbname=NULL, dbpath=NULL,
            extraArgs='', createDirectory=FALSE,
            verbose=TRUE)

Arguments

seqs  Sequence(s) to create a BLAST database from. This can be either an XStringSet or a path to a FASTA file.
dbtype  Either 'prot' for amino acid input, or 'nucl' for nucleotide input.
dbname  Name of the resulting database. If not provided, defaults to a random string prefixed by blastdb.
dbpath  Path where database should be created. If not provided, defaults to TMPDIR.
extraArgs  Additional arguments to be passed to the query executed on the command line. This should be a single character string.
createDirectory  Should a directory be created for the database if it doesn’t exist? If FALSE, the function will throw an error instead of creating a directory.
verbose  Should output be displayed?

Details

This offers a quick way to create BLAST databases from R. This function essentially wraps the makeblastdb commandline function. All arguments supported by makeblastdb are supported in the extraArgs argument.
### Value

Returns a length 2 named character vector specifying the name of the BLAST database and the path to it.

### Author(s)

Aidan Lakshman <ahl27@pitt.edu>

### See Also

BlastSeqs

### Examples

```r
#
```

### Description

Calculates Moran’s I to measure spatial autocorrelation for a set of signals dispersed in space.

### Usage

```r
MoransI(values, weights, alternative='two.sided')
```

### Arguments

- **values**: Numeric vector containing signals for each point in space.
- **weights**: Distances between each point in space. This should be a numeric object of class `dist` with `Size` attribute equivalent to the length of `values`.
- **alternative**: For hypothesis testing against the null of no spatial correlation, how should a p-value be calculated? Should be one of c("two.sided", "less", "greater"), or an unambiguous abbreviation.

### Details

Moran’s I is a measure of how much the spatial arrangement of a set of datapoints correlates with the value of each datapoint. The index takes a value in the range $[-1, 1]$, with values close to 1 indicating high correlation between location and value (points have increasingly similar values as they increase in proximity), values close to -1 indicating anticorrelation (points have increasingly different values as they increase in proximity), and values close to 0 indicating no correlation. The value itself is calculated as:

$$ I = \frac{N}{W} \sum_{i}^{N} \sum_{j}^{N} w_{ij}(x_i - \bar{x})(x_j - \bar{x}) \sum_{i}^{N}(x_i - \bar{x})^2 $$
Here, \( N \) is the number of points, \( w_{ij} \) is the distance between points \( i \) and \( j \), \( W = \sum_{i,j} w_{ij} \) (the sum of all the weights), \( x_i \) is the value of point \( i \), and \( \bar{x} \) is the sample mean of the values.

Moran’s \( I \) has a closed form calculation for variance and expected value, which are calculated within this function. The full form of the variance is fairly complex, but all the equations are available for reference here.

A p-value is estimated using the expected value and variance using a null hypothesis of no spatial autocorrelation, and the alternative hypothesis specified in the \texttt{alternative} argument. Note that if fewer than four datapoints are supplied, the variance of Moran’s \( I \) is infinite. The function will return a standard deviation of \texttt{Inf} and a p-value of 1 in this case.

\textbf{Value}

A \texttt{list} object containing the following named values:

- \texttt{observed}: The value of Moran’s \( I \) (numeric in the range \([-1, 1]\)).
- \texttt{expected}: The expected value of Moran’s \( I \) for the input data.
- \texttt{sd}: The standard deviation of Moran’s \( I \) for the input data.
- \texttt{p.value}: The p-value for the input data, calculated with the alternative hypothesis as specified in \texttt{alternative}.

\textbf{Author(s)}

Aidan Lakshman <ahl27@pitt.edu>

\textbf{References}


\textbf{Examples}

```r
# Make a distance matrix for a set of 50 points
# These are just random numbers in the range [0.1,2]
NUM_POINTS <- 50
distmat <- as.dist(matrix(runif(NUM_POINTS**2, 0.1, 2),
                         ncol=NUM_POINTS))

# Generate some random values for each of the points
vals <- runif(NUM_POINTS, 0, 3)

# Calculate Moran's I
MoransI(vals, distmat, alternative='two.sided')

# effect size should be pretty small
# and p-value close to 0.5
# since this is basically random data
```
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
```r
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
```r
PairSummaries(SyntenyLinks,
DBPATH,
PIIDs = 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.
**PairSummaries**

**Storage**

Numerical 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**

```r
# this function will be deprecated soon,
```
PhyloDistance

# please see the new SummarizePairs() function.
DBPATH <- system.file("extdata",
        "Endosymbionts_v02.sqlite",
        package = "SynExtend")

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

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

PhyloDistance

| PhyloDistance | Calculate Distance between Unrooted Phylogenies |

Description

Calculates distance between two unrooted phylogenies using a variety of metrics.

Usage

PhyloDistance(dend1, dend2,
               Method=c("CI", "RF", "KF", "JRF"),
               RawScore=FALSE, JRFExp=2)

Arguments

dend1 An object of class dendrogram, representing an unrooted bifurcating phylogenetic tree.
dend2 An object of class dendrogram, representing an unrooted bifurcating phylogenetic tree.
Method Method to use for calculating tree distances. The following values are supported: "CI", "RF", "KF", "JRF". See Details for more information.
RawScore If FALSE, returns distance between the two trees. If TRUE, returns the component values used to calculate the distance. This may be preferred for methods like GRF. See the pages specific to each algorithm for more information on what values are reported.
JRFExp k-value used in calculation of JRF Distance. Unused if Method is not "JRF".

Details

This function implements a variety of tree distances, specified by the value of Method. The following values are supported, along with links to documentation pages for each function:

- "RF": Robinson-Foulds Distance
- "CI": Clustering Information Distance
- "JRF": Jaccard-Robinson-Foulds Distance, equivalent to the Nye Distance Metric when JRFVal=1
• "KF": Kuhner-Felsenstein Distance

Information on each of these algorithms, how scores are calculated, and references to literature can be found at the above links. Method "CI" is selected by default due to recent work showing this method as the most robust tree distance metric under general conditions.

**Value**

Returns a normalized distance, with 0 indicating identical trees and 1 indicating maximal difference. If the trees have no leaves in common, the function will return 1.

If RawScore=TRUE, returns a vector of the components used to calculate the distance. This is typically a length 3 vector, but specific details can be found on the description for each algorithm.

**Note**

Note that this function requires the input dendrograms to be labeled alike (ex. leaf labeled abc in dend1 represents the same species as leaf labeled abc in dend2). Labels can easily be modified using **dendrapply**.

**Author(s)**

Aidan Lakshman <ahl27@pitt.edu>

**See Also**

Robinson-Foulds Distance
Clustering Information Distance
Jaccard-Robinson-Foulds Distance
Kuhner-Felsenstein Distance

**Examples**

```r
# making some toy dendrograms
set.seed(123)
dm1 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
dm2 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))

tree1 <- as.dendrogram(hclust(dm1))
tree2 <- as.dendrogram(hclust(dm2))

# Robinson-Foulds Distance
PhyloDistance(tree1, tree2, Method="RF")

# Clustering Information Distance
PhyloDistance(tree1, tree2, Method="CI")

# Kuhner-Felsenstein Distance
PhyloDistance(tree1, tree2, Method="KF")

# Nye Distance Metric
```
PhyloDistance(CIDist)

PhyloDistance(tree1, tree2, Method="JRF", JRFExp=1)

# Jaccard-Robinson-Foulds Distance
PhyloDistance(tree1, tree2, Method="JRF", JRFExp=2)

Description

Calculate distance between two unrooted phylogenies using mutual clustering information of branch partitions.

Details

This function is called as part of PhyloDistance and calculates tree distance using the clustering information approach first described in Smith (2020). This function iteratively pairs internal tree branches of a phylogeny based on their similarity, then scores overall similarity as the sum of these measures. The similarity score is then converted to a distance by normalizing by the average entropy of the two trees. This metric has been demonstrated to outperform numerous other metrics in capabilities; see the original publication cited in References for more information.

Users may wish to use the actual similarity values rather than a distance metric; the option to specify RawScore=TRUE is provided for this case. Distance is calculated as \( \frac{M-S}{M} \), where \( M = \frac{1}{2}(H_1 + H_2) \), \( H_i \) is the entropy of the \( i \)'th tree, and \( S \) is the similarity score between them. As shown in the original publication, this satisfies the necessary requirements to be considered a distance metric. Setting RawScore=TRUE will instead return a vector with \( (S, H_1, H_2, p) \), where \( p \) is an approximation for the two sided p-value of the result based on random simulations from Smith (2020).

Value

Returns a normalized distance, with 0 indicating identical trees and 1 indicating maximal difference. Note that branch lengths are not considered, so two trees with different branch lengths may return a distance of 0.

If RawScore=TRUE, returns a named length 4 vector with the first entry the similarity score, subsequent entries the entropy values for each tree, and the last entry the approximate p-value for the result based on simulations.

If the trees have no leaves in common, the function will return 1 if RawScore=FALSE, and c(0, NA, NA, NA) if TRUE.

Note

Note that this function requires the input dendrograms to be labeled alike (ex. leaf labeled abc in dend1 represents the same species as leaf labeled abc in dend2). Labels can easily be modified using dendrapply.
Author(s)
Aidan Lakshman <ahl27@pitt.edu>

References

Examples
# making some toy dendrograms
set.seed(123)
dm1 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
dm2 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
tree1 <- as.dendrogram(hclust(dm1))
tree2 <- as.dendrogram(hclust(dm2))

# get RF distance
PhyloDistance(tree1, tree2, Method="CI")

# get similarity score with individual entropies
PhyloDistance(tree1, tree2, Method="CI", RawScore=TRUE)

---

PhyloDistance-JRFDist  Jaccard-Robinson-Foulds Distance

Description
Calculate JRF distance between two unrooted phylogenies.

Details
This function is called as part of PhyloDistance and calculates the Jaccard-Robinson-Foulds distance between two unrooted phylogenies. Each dendrogram is first pruned to only internal branches implying a partition in the shared leaf set; trivial partitions (where one leaf set contains 1 or 0 leaves) are ignored.

The total score is calculated by pairing branches and scoring their similarity. For a set of two branches $A, B$ that partition the leaves into $(A_1, A_2)$ and $(B_1, B_2)$ (resp.), the distance between the branches is calculated as:

$$2 - 2 \left( \frac{|X \cap Y|}{|X \cup Y|} \right)^k$$

where $X \in (A_1, A_2), Y \in (B_1, B_2)$ are chosen to maximize the score of the pairing, and $k$ the value of ExpVal. The sum of these scores for all branches produces the overall distance between the two trees, which is then normalized by the number of branches in each tree.
There are a few special cases to this distance. If ExpVal=1, the distance is equivalent to the metric introduced in Nye et al. (2006). As ExpVal approaches infinity, the value becomes close to the (non-Generalized) Robinson Foulds Distance.

**Value**

Returns a normalized distance, with 0 indicating identical trees and 1 indicating maximal difference. If RawScore=TRUE, returns a named length 3 vector with the first entry the summed distance score over the branch pairings, and the subsequent entries the number of partitions for each tree. If the trees have no leaves in common, the function will return 1 if RawScore=FALSE, and c(0, NA, NA) if TRUE.

**Note**

Note that this function requires the input dendrograms to be labeled alike (ex. leaf labeled abc in dend1 represents the same species as leaf labeled abc in dend2). Labels can easily be modified using `dendrapply`.

**Author(s)**

Aidan Lakshman <ahl27@pitt.edu>

**References**


**Examples**

# making some toy dendrograms
set.seed(123)
dm1 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
dm2 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
tree1 <- as.dendrogram(hclust(dm1))
tree2 <- as.dendrogram(hclust(dm2))

# Nye Metric
PhyloDistance(tree1, tree2, Method="JRF", JRFExp=1)

# Jaccard-RobinsonFoulds
PhyloDistance(tree1, tree2, Method="JRF", JRFExp=2)

# Good approximation to RF Dist (note RFDist is much faster for this)
PhyloDistance(tree1, tree2, Method="JRF", JRFExp=1000)
PhyloDistance(tree1, tree2, Method="RF")
Description

Calculate KF distance between two unrooted phylogenies.

Details

This function is called as part of PhyloDistance and calculates Kuhner-Felsenstein distance between two unrooted phylogenies. Each dendrogram is first pruned to only internal branches implying a partition in the shared leaf set; trivial partitions (where one leaf set contains 1 or 0 leaves) are ignored. The total score is calculated as the sum of squared differences between lengths of branches implying equivalent partitions. If a particular branch is unique to a given tree, it is treated as having length 0 in the other tree. The final score is normalized by the sum of squared lengths of all internal branches of both trees, resulting in a final distance that ranges from 0 to 1.

Value

Returns a normalized distance, with 0 indicating identical trees and 1 indicating maximal difference. If the trees have no leaves in common, the function will return 1.

Note

Note that this function requires the input dendrograms to be labeled alike (ex. leaf labeled abc in dend1 represents the same species as leaf labeled abc in dend2). Labels can easily be modified using dendrapply.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

References


Examples

# making some toy dendrograms
set.seed(123)
dm1 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
dm2 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
tree1 <- as.dendrogram(hclust(dm1))
tree2 <- as.dendrogram(hclust(dm2))
PhyloDistance-RFDist

# get KF distance
PhyloDistance(tree1, tree2, Method="KF")

PhyloDistance-RFDist  Robinson-Foulds Distance

Description

Calculate RF distance between two unrooted phylogenies.

Details

This function is called as part of PhyloDistance and calculates Robinson-Foulds distance between two unrooted phylogenies. Each dendrogram is first pruned to only internal branches implying a partition in the shared leaf set; trivial partitions (where one leaf set contains 1 or 0 leaves) are ignored. The total score is calculated as the number of unique partitions divided by the total number of partitions in both trees. Setting RawScore=TRUE will instead return a vector with \( (P_{\text{shared}}, P_1, P_2) \), corresponding to the shared partitions and partitions in the first and second trees (respectively).

This algorithm incorporates some optimizations from Pattengale et al. (2007) to improve computation time of the original fast RF algorithm detailed in Day (1985).

Value

Returns a normalized distance, with 0 indicating identical trees and 1 indicating maximal difference. Note that branch lengths are not considered, so two trees with different branch lengths may return a distance of 0.

If RawScore=TRUE, returns a named length 3 vector with the first entry the number of unique partitions, and the subsequent entries the number of partitions for each tree.

If the trees have no leaves in common, the function will return 1 if RawScore=FALSE, and c(0, NA, NA) if TRUE.

Note

Note that this function requires the input dendrograms to be labeled alike (ex. leaf labeled abc in dend1 represents the same species as leaf labeled abc in dend2). Labels can easily be modified using dendrapply.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>
References


Examples

# making some toy dendrograms
set.seed(123)
dm1 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))
dm2 <- as.dist(matrix(runif(64, 0.5, 5), ncol=8))

tree1 <- as.dendrogram(hclust(dm1))
tree2 <- as.dendrogram(hclust(dm2))

# get RF distance
PhyloDistance(tree1, tree2, Method="RF")

# get number of unique splits per tree
PhyloDistance(tree1, tree2, Method="RF", RawScore=TRUE)

plot.EvoWeb

Plot predictions in a EvoWeb object

Description

EvoWeb objects are outputted from `predict.EvoWeaver`.

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

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

Arguments

- **x**
  A EvoWeb object. See EvoWeb

- **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 EvoWeb 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.EvoWeaver
- EvoWeb
**Examples**

```r
exData <- get(data("ExampleStreptomycesData"))
ew <- EvoWeaver(exData$Genes)

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

plot(evoweb)
```

---

**predict.EvoWeaver  Make predictions with EvoWeaver objects**

**Description**

This S3 method predicts pairwise functional associations between gene groups encoded in a `EvoWeaver` object. This returns an object of type `EvoWeb`, which is essentially an adjacency matrix with some extra S3 methods to make printing cleaner.

**Usage**

```r
## S3 method for class 'EvoWeaver'
predict(object, Method='Ensemble',
         Subset=NULL, Processors=1L,
         MySpeciesTree=SpeciesTree(object),
         PretrainedModel=NULL,
         NoPrediction=FALSE,
         ReturnRawData=FALSE, Verbose=TRUE, ...)
```

**Arguments**

- **object**: A `EvoWeaver` object
- **Method**: Method(s) to use for prediction. This can be a character vector with multiple entries for predicting using multiple methods. See 'Details' for more information.
- **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 specified by name (character) or by index (numeric).
- **Processors**: Number of cores to use for methods that support multithreaded execution. Setting to `NULL` or a negative value will use the value of `detectCores()`, or one core if the number of available cores cannot be determined. See Note for more information.
**predict.EvoWeaver**

Phylogenetic tree of all genomes in the dataset. Required for `Method = c('ContextTree', 'GainLoss', 'CorrGL', 'ColocMoran', 'Behdenna').` 'Behdenna' requires a rooted, bifurcating tree (other values of `Method` can handle arbitrary trees).

Note that `EvoWeaver` can automatically infer a species tree if initialized with dendrogram objects.

**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'.`

**NoPrediction**

For `Method = 'Ensemble',` should data be returned prior to making predictions? If `TRUE`, this will instead return a data frame object with predictions from each algorithm for each pair. This data frame is typically used to train an ensemble model.

If `FALSE`, `EvoWeaver` will return predictions for each pair (using user model if provided or a built-in otherwise).

**ReturnRawData**

Internal parameter used for ensemble predictions. Should not be set by the user.

**Verbose**

Logical indicating whether to print progress bars and messages. Defaults to `TRUE`.

**...**

Additional parameters for other predictors and consistency with generic.

**Details**

`predict.EvoWeaver` wraps several methods to create an easy interface for multiple prediction types. `Method = 'Ensemble'` is the default value, but each of the component analyses can also be accessed. The following is a list of all algorithms implemented in `EvoWeaver` (* denotes algorithms used in the `EvoWeaver` publication):

- 'Ensemble': Ensemble prediction combining individual coevolutionary predictors. See Note below.
- * 'Jaccard': Jaccard distance of Presence/Absence (P/A) profiles
- 'Hamming': Hamming distance of P/A profiles
- * 'MutualInformation': MI of P/A profiles
- * 'PAPV': 1 - p_value of P/A profiles
- 'ProfDCA': Direct Coupling Analysis of P/A profiles
- 'Behdenna': Analysis of Gain/Loss events following Behdenna et al. (2016)
- * 'CorrGL': Correlation of ancestral Gain/Loss events
- * 'GainLoss': Score-based method based on distance between inferred ancestral Gain/Loss events
- * 'MirrorTree': MirrorTree using Random Projection for dimensionality reduction
- * 'ContextTree': MirrorTree with Random Projection correcting for species tree and P/A conservation
- * 'Coloc': Co-localization analysis
• *'ColocMoran': Co-localization analysis using Moran’s I for phylogenetic correction and significance
• *'TranscripMI': Mutual Information of Transcriptional Direction
• *'NVDI': Correlation of distribution of sequence level residues following Zhao et al. (2022)
• *'ResidueMI': Mutual information of sites in multiple sequence alignment

The best performing individual predictors are c('CorrGL', 'GainLoss', 'MirrorTree', 'Jaccard'). Users interested in running quick analyses should use c('CorrGL', 'GainLoss', 'Jaccard'). Additional information and references for each prediction algorithm can be found at the following pages:

• EvoWeaver Phylogenetic Profiling Methods
• EvoWeaver Phylogenetic Structure Methods
• EvoWeaver Gene Organization Methods
• EvoWeaver Sequence-Level Methods

This returns a EvoWeb object, an S3 class that makes formatting and printing of results slightly nicer. See EvoWeb for more information.

Different methods require different types of input. The constructor EvoWeaver will notify the user which methods are runnable with the given data. Method Ensemble automatically selects the methods that can be run with the given input data. See EvoWeaver for more information on input data types.

Value

Returns a EvoWeb object. See EvoWeb for more info.

Note

The current ensemble method included with EvoWeaver is out of date. EvoWeaver’s publication used a random forest model from the randomForest package for prediction. The next release of EvoWeaver will include multiple new built-in ensemble methods, but in the interim users are recommended to rely on randomForest or neuralnet. Planned algorithms are random forests and feed-forward neural networks. Feel free to contact me regarding other models you would like to see added.

If NumCores is set to NULL, EvoWeaver will use one less core than is detected, or one 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.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>
See Also

EvoWeaver
EvoWeb
EvoWeaver Phylogenetic Profiling Predictors
EvoWeaver Phylogenetic Structure Predictors
EvoWeaver Gene Organization Predictors
EvoWeaver Sequence-Level Predictors

Examples

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

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

# Subset isn’t necessary but is faster for a working example
evoweb1 <- predict(ew, Subset=1:10, MySpeciesTree=exData$Tree)

# print out results as an adjacency matrix
evoweb1

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

datavals <- predict(ew, 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')

testEvoWeaverObject <- EvoWeaver(exData$Genes[51:60])
evoweb2 <- predict(testEvoWeaverObject,
    PretrainedModel=myModel)

# Print result as a matrix of pairwise scores
evoweb2

PrepareSeqs

Return gene sequences.
PrepareSeqs

Description
Given a SynExtend object with a GeneCalls attribute, and a DECIPHER database, return all gene sequences and their translations.

Usage
PrepareSeqs(SynExtendObject, DataBase, DefaultTranslationTable = "11", Identifiers = NULL, Storage = 1, Verbose = FALSE)

Arguments
SynExtendObject
An object of class PairSummaries or of LinkedPairs. Object must have a GeneCalls attribute.

DataBase
A character string pointing to a SQLite database, or a connection to a DECIPHER database.

DefaultTranslationTable
A character vector of length 1 identifying the translation table to use if one is not supplied in the GeneCalls attribute.

Identifiers
By default NULL, but can be used to supply a vector of character identifiers for returning a subset of prepared sequences.

Storage
A soft memory limit for how much space to allow when building the resulting object. Translated to Gb.

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

Details
PrepareSeqs returns the sequences of genes and their translations where appropriate.

Value
An object of class FeatureSeqs.

Author(s)
Nicholas Cooley <npc19@pitt.edu>

See Also
SummarizePairs, NucleotideOverlap, FindSyteny
SelectByK

Examples

```r
DBPATH <- system.file("extdata",  # Examples
  "Endosymbionts_v02.sqlite",
  package = "SynExtend")

data("Endosymbionts_LinkedFeatures", package = "SynExtend")
CurrentSeqs <- PrepareSeqs(SynExtendObject = Endosymbionts_LinkedFeatures,
  DataBase = DBPATH,
  Verbose = TRUE)
```

---

SelectByK  

Predicted pair trimming using K-means.

Description

A relatively simple k-means clustering approach to drop predicted pairs that belong to clusters with a PID centroid below a specified user threshold.

Usage

```r
SelectByK(Pairs,
  UserConfidence = 0.5,
  ClusterScalar = 1,
  MaxClusters = 15L,
  ReturnAllCommunities = FALSE,
  Verbose = FALSE,
  ShowPlot = FALSE,
  RetainHighest = TRUE)
```

Arguments

- **Pairs**: An object of class `PairSummaries`.
- **UserConfidence**: A numeric value greater than 0 and less than 1 that represents a minimum PID centroid that users believe represents a TRUE predicted pair.
- **ClusterScalar**: A numeric value used to scale selection of how many clusters are used in kmeans clustering. Total within-cluster sum of squares are fit to a right hyperbola, and the half-max is used to select cluster number. “ClusterScalar” is multiplied by the half-max to adjust cluster number selection.
- **MaxClusters**: Integer value indicating the largest number of clusters to test in a series of k-means clustering tests.
- **ReturnAllCommunities**: A logical value. If “TRUE”, function returns a list where the second position is a list of “PairSummaries” tables for each k-means cluster. By default is “FALSE”, returning only a “PairSummaries” object of the retained predicted pairs.
ShowPlot Logical indicating whether or not to plot the CDFs for the PIDs of all k-means clusters for the determined cluster number.

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

RetainHighest Logical indicating whether to retain the cluster with the highest PID centroid in the case where the PID is below the specified user confidence.

Details

SelectByK uses a naive k-means routine to select for predicted pairs that belong to clusters whose centroids are greater than or equal to the user specified PID confidence. This means that the confidence is not a minimum, and that pairs with PIDs below the user confidence can be retained. The sum of within cluster sum of squares is used to approximate “knee” selection with the user supplied “ClusterScalar” value. By default, with a “ClusterScalar” value of 1 the half-max of a right-hyperbola fitted to the sum of within-cluster sum of squares is used to pick the cluster number for evaluation. “ClusterScalar” is multiplied by the half-max to tune cluster number selection. This function is intended to be used at the genome-to-genome comparison level, and not say, at the level of an all-vs-all comparison of many genomes.

Value

An object of class PairSummaries.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

PairSummaries, NucleotideOverlap, link{SubSetPairs}, FindSynteny

Examples

# this function will be deprecated soon,
# please see the new ClusterByK() function.

DBPATH <- system.file("extdata",
  "Endosymbions_v02.sqlite",
  package = "SynExtend")

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

Pairs <- PairSummaries(SyntenyLinks = Endosymbions_LinkedFeatures,
  PIDs = TRUE,
  Score = TRUE,
  DBPATH = DBPATH,
  Verbose = TRUE)

Pairs02 <- SelectByK(Pairs = Pairs)
SequenceSimilarity

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

Description

Takes in a DNAStringSet or AAStringSet representing a pairwise alignment and a substitution 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 substitution 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

```r
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)
```

---

### simMat

#### Similarity Matrices

**Description**

The `simMat` object is an internally utilized class that provides similar functionality to the `dist` object, but with matrix-like accessors.

Like `dist`, this object stores values as a vector, reducing memory by making use of assumed symmetry. `simMat` currently only supports numeric data types.

**Usage**

```r
## Create a blank sym object
simMat(VALUE, nelem, NAMES=NULL, DIAG=FALSE)

## S3 method for class 'vector'
as.simMat(x, NAMES=NULL, DIAG=TRUE, ...)

## S3 method for class 'matrix'
as.simMat(x, ...)

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

## S3 method for class 'simMat'
as.matrix(x, ...)

## S3 method for class 'simMat'
```
as.data.frame(x, ...)

## S3 method for class 'simMat'
Diag(x, ...)

## S3 replacement method for class 'simMat'
Diag(x) <- value

Arguments

VALUE Numeric (or NA_real_) indicating placeholder values. A vector of values can be provided for this function if desired.

nelem Integer; number of elements represented in the matrix. This corresponds to the number of rows and columns of the object, so setting nelem=10 would produce a 10x10 matrix.

NAMES Character (Optional); names for each row/column. If provided, this should be a character vector of length equal to nelem.

DIAG Logical; Is the diagonal included in the data? If FALSE, the constructor generates 1s for the diagonal.

x Various; for print and Diag, the "simMat" object to print. For as.vector or as.matrix, the vector or matrix (respectively). Note that as.matrix expects a symmetric matrix–providing a non-symmetric matrix will take only the upper triangle and produce a warning.

value Numeric; value(s) to replace diagonal with.

... Additional parameters provided for consistency with generic.

Details

The simMat object has a very similar format to dist objects, but with a few notable changes:

- simMat objects have streamlined print and show methods to make displaying large matrices better. print accepts an additional argument n corresponding to the maximum number of rows/columns to print before truncating.
- simMat objects support matrix-style get/set operations like s[1,] or s[1,3:5]
- simMat objects allow any values on the diagonal, rather than just zeros as in dist objects.
- simMat objects support conversion to matrices and data.frame objects
- simMat objects implement get/set Diag() methods. Note usage of capitalized Diag; this is to avoid conflicts and weirdness with using base diag.

See the examples for details on using these features.

The number of elements printed when calling print or show on a simMat object is determined by the "SynExtend.simMat" option.
simMat and as.simMat return an object of class "simMat". Internally, the object stores the upper triangle of the matrix similar to how dist stores objects.

The object has the following attributes (besides "class" equal to "simMat"):

- **nrow**: the number of rows in the matrix implied by the vector
- **NAMES**: the names of the rows/columns

as.matrix(s) returns the equivalent matrix to a "simMat" object.

as.data.frame(s) returns a data.frame object corresponding to pairwise similarities.

**Author(s)**

Aidan Lakshman <ahl27@pitt.edu>

**Examples**

```
## Creating a blank simMat object initialized to zeros
s <- simMat(0, nelem=20)
s
## Print out 5 rows instead of 10
print(s, n=5)

## Create a simMat object with 5 entries from a vector
dimn <- 5
vec <- 1:(dimn*(dimn-1) / 2)
s1 <- as.simMat(vec, DIAG=FALSE)
s1

## Here we include the diagonal
vec <- 1:(dimn*(dimn+1) / 2)
s2 <- as.simMat(vec, DIAG=TRUE)
s2

## Subsetting
s2[1,]
s2[1,3:4]
# all entries except first row
s2[-1,]
# all combos not including 1
s2[-1,-1]

## Replace values (automatically recycled)
s2[1,] <- 10
s2

## Get/set diagonal
Diag(s1)
Diag(s1) <- 5
s1
```
**subset.dendrogram**  
*Subsetting dendrogram objects*

**Description**
Subsets dendrogram objects based on leaf labels. Subsetting can either be by leaves to keep, or leaves to remove.

NOTE: This man page is specifically for subset.dendrogram, see ?base::subset for the generic subset function defined for vectors, matrices, and data frames.

**Usage**
```r
## S3 method for class 'dendrogram'
subset(x, subset, invert=FALSE, ...)```

**Arguments**
- `x` An object of class 'dendrogram'
- `subset` A vector of labels to keep (see `invert`).
- `invert` If set to TRUE, subset to only the leaves not in `subset`.
- `...` Additional arguments for consistency with generic.

**Value**
An object of class 'dendrogram' corresponding to the subsetted tree.

**Note**
If none of the labels specified in the `subset` argument appear in the tree (or if all do when `invert=TRUE`), a warning is thrown and an empty object of class 'dendrogram' is returned.

**Author(s)**
Aidan Lakshman <ahl27@pitt.edu>

**See Also**
- `subset`
**Examples**

```r
# Subset to first 10 labels
d1 <- subset(d, labels(d)[1:10])
plot(d1)
# Subset d1 to all except the first 2 labels
d2 <- subset(d1, labels(d1)[1:2], invert=TRUE)
plot(d2)
```

---

**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**

```r
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”.
**SummarizePairs**

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**

# expected to be deprecated soon...
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", "TotalMatch")
# remove pairs only based on user defined thresholds
Pairs3 <- SubSetPairs(CurrentPairs = Endosymbionts_Pairs03,
    UserThresholds = THRESH,
    RejectCompetitors = FALSE,
    Verbose = TRUE)

---

**SummarizePairs**

*Provide summaries of hypothetical orthologs.*

**Description**

Given the correct set of SynExtend objects and a DECIPHER database, return a data.frame of summarized genomic feature pairs. SummarizePairs will collect all the linked genomic features in the supplied **LinkedPairs-class** object and return descriptions of the alignments of those features.
Usage

SummarizePairs(SynExtendObject,
   FeatureSeqs,
   DataBase,
   AlignmentFun = "AlignProfiles",
   RetainAnchors = FALSE,
   DefaultTranslationTable = "11",
   KmerSize = 5,
   IgnoreDefaultStringSet = FALSE,
   Verbose = FALSE,
   ShowPlot = FALSE,
   Processors = 1,
   ...)

Arguments

SynExtendObject
   An object of class LinkedPairs-class.
FeatureSeqs
   An object of class FeatureSeqs.
DataBase
   A character string pointing to a SQLite database, or a connection to a DECIPHER database.
AlignmentFun
   A character string specifying a link{DECIPHER} alignment function. Currently only supports AlignProfiles and AlignPairs.
RetainAnchors
   An argument that only affects AlignPairs; provide the kmer hits supplied by FindSynteny as alignment anchors.
DefaultTranslationTable
   A character vector of length 1 identifying the translation table to use if one is not supplied in the GeneCalls attribute.
KmerSize
   An integer specifying what Kmer size to collect Kmer distance between sequences at.
IgnoreDefaultStringSet
   A soft memory limit for how much space to allow when building the resulting object. Translated to Gb.
Verbose
   Logical indicating whether or not to display a progress bar and print the time difference upon completion.
ShowPlot
   Logical indicating whether or not to provide a plot of features collected by the function.
Processors
   An integer value indicating how many processors to supply to AlignPairs.
...

Details

SummarizePairs collects features describing each linked feature pair. These include an alignment PID, an alignment Score, a Kmer distance, a concensus score for the linking hits –or whether or not linking hits are in similar places in each feature– and a few other features.
SuperTree

Value
An object of class PairSummaries.

Author(s)
Nicholas Cooley <npc19@pitt.edu>

See Also
PrepareSeqs, NucleotideOverlap, FindSynteny, LinkedPairs-class

Examples

```r
DBPATH <- system.file("extdata", "Endosymbionts_v02.sqlite", package = "SynExtend")
data("Endosymbionts_LinkedFeatures", package = "SynExtend")
Endosymbiont_Seqs <- PrepareSeqs(SynExtendObject = Endosymbionts_LinkedFeatures, DataBase = DBPATH, Verbose = TRUE)
SummarizedPairs <- SummarizePairs(SynExtendObject = Endosymbionts_LinkedFeatures, FeatureSeqs = Endosymbiont_Seqs, DataBase = DBPATH)
```

SuperTree Create a Species Tree from Gene Trees

Description
Given a set of unrooted gene trees, creates a species tree. This function works for rooted gene trees, but may not accurately root the resulting tree.

Usage

```r
SuperTree(myDendList, NAMEFUN=NULL, Verbose=TRUE, Processors=1)
```

Arguments

- `myDendList`: A list of dendrogram objects, where each entry is an unrooted gene tree.
- `NAMEFUN`: Optional input specifying a function to apply to each leaf to convert gene tree leaf labels into species names. This function should take as input a character vector and return a character vector of the same size. By default equals NULL, indicating that gene tree leaves are already labeled with species identifiers. See details for more information.
- `Verbose`: Should output be displayed?
- `Processors`: Number of processors to use for calculating the final species tree.
Details

This implementation follows the ASTRID algorithm for estimating a species tree from a set of unrooted gene trees. Input gene trees are not required to have identical species sets, as the algorithm can handle missing entries in gene trees. The algorithm essentially works by averaging the Cophenetic distance matrices of all gene trees, then constructing a neighbor-joining tree from the resulting distance matrix. See the original paper linked in the references section for more information.

If two species never appear together in a gene tree, their distance cannot be estimated in the algorithm and will thus be missing. SuperTree handles this by imputing the value using the distances available with data-interpolating empirical orthogonal functions (DINEOF). This approach has relatively high accuracy even up to high levels of missingness. Eigenvector calculation speed is improved using a Lanczos algorithm for matrix compression.

SuperTree allows an optional argument called `NAMEFUN` to apply a renaming step to leaf labels. Gene trees as constructed by other functions in SynExtend (ex. `DisjointSet`) often include other information aside from species name when labeling genes, but SuperTree requires that leaf nodes of the gene tree are labeled with just an identifier corresponding to which species/genome each leaf is from. Duplicate values are allowed. See the examples section for more details on what this looks like and how to handle it.

Value

A dendrogram object corresponding to the species tree constructed from input gene trees.

Author(s)

Aidan Lakshman <ahl27@pitt.edu>

References


See Also

TreeLine, SuperTreeEx

Examples

# Loads a list of dendrograms
# each is a gene tree from Streptomyces genomes
data("SuperTreeEx", package="SynExtend")

# Notice that the labels of the tree are in #_#_# format
# See the man page for SuperTreeEx for more info
labs <- labels(exData[[1]])
labs
# The first number corresponds to the species,  
# so we need to trim the rest in each leaf label
namefun <- function(x) gsub("([0-9A-Za-z]+.)", "\1", x)  
namefun(labs) # trims to just first number

# This function replaces gene identifiers with species identifiers  
# we pass it to NAMEFUN
# Note NAMEFUN should take in a character vector and return a character vector
tree <- SuperTree(exData, NAMEFUN=namefun)

---

**SuperTreeEx**  
**Example Dendrograms**

### Description
A set of 20 dendrograms for use in `SuperTree` examples.

### Usage

```r
data("SuperTreeEx")
```

### Value
A list with 20 elements, where each is a object of type `dendrogram` corresponding to a gene tree constructed from a set of 301 *Streptomyces* genomes. Each leaf node is labeled in the form A_B_C, where A is a number identifying the genome, B is a number identifying the contig, and C is a number identifying the gene. Altogether, each label uniquely identifies a gene.

### Examples

```r
data(SuperTreeEx, package="SynExtend")
```
Index

* GeneCalls
  gffToDataFrame, 48
* datasets
  BuiltInEnsembles, 8
  CIDist_NullDist, 9
  Endosymbionts_GeneCalls, 17
  Endosymbionts_LinkedFeatures, 17
  Endosymbionts_Pairs01, 18
  Endosymbionts_Pairs02, 18
  Endosymbionts_Pairs03, 19
  Endosymbionts_Sets, 19
  Endosymbionts_Synteny, 20
  ExampleStreptomycesData, 36
  Generic, 47
  SuperTreeEx, 83
  [.LinkedPairs (LinkedPairs), 49

AlignPairs, 80
AlignProfiles, 74, 80
AlignSeqs, 74
AlignTranslation, 74
Ancestral.EvoWeaver
  (EvoWeaver-SLPreds), 33
as.data.frame.simMat (simMat), 74
dendrogram, 13
as.matrix.simMat (simMat), 74
as.simMat (simMat), 74
attributes, 12

Behdenna.EvoWeaver (EvoWeaver-PPPreds), 29
BlastSeqs, 3, 51
BlockExpansion, 4
BlockReconciliation, 6
BuiltInEnsembles, 8, 26, 67

CIDist (PhyloDistance-CIDist), 59
CIDist_NullDist, 9
ClusterByK, 10

Clustering Information Distance, 9, 32,
57, 58
Coloc.EvoWeaver (EvoWeaver-GOPreds), 28
ColocMoran.EvoWeaver
  (EvoWeaver-GOPreds), 28
ContextTree.EvoWeaver
  (EvoWeaver-PPPreds), 31
CorrGL.EvoWeaver (EvoWeaver-PPPreds), 29
data.frame, 4, 38, 67
dendrapply, 11, 58, 59, 61–63
dendrogram, 12, 15, 82, 83
Diag (simMat), 74
Diag <- (simMat), 74
DisjointSet, 14, 42, 82
dist, 51, 74
DistanceMatrix, 74
DPhyloStatistic, 15

Endosymbionts_GeneCalls, 17
Endosymbionts_LinkedFeatures, 17
Endosymbionts_Pairs01, 18
Endosymbionts_Pairs02, 18
Endosymbionts_Pairs03, 19
Endosymbionts_Sets, 19
Endosymbionts_Synteny, 20
EstimateExoLabel, 20, 39
EstimateRearrangementScenarios
  (EstimRearrScen), 22
EstimRearrScen, 22
EvoWeaver, 25, 28–36, 66, 68, 69
EvoWeaver Gene Organization Methods, 68
EvoWeaver Gene Organization
  Predictors, 31, 33, 34, 69
EvoWeaver Phylogenetic Profiling
  Methods, 68
EvoWeaver Phylogenetic Profiling
  Predictors, 29, 33, 34, 69
EvoWeaver Phylogenetic Structure
  Methods, 68
EvoWeaver Phylogenetic Structure Predictors, 29, 31, 34, 69
EvoWeaver Sequence-Level Methods, 68
EvoWeaver Sequence-Level Predictors, 29, 31, 33, 69
EvoWeaver-class (EvoWeaver), 25
EvoWeaver-PPPreds, 28
EvoWeaver-PPPreds, 29
EvoWeaver-PSPreds, 31
EvoWeaver-SLPreds, 33
EvoWeaver-utils (EvoWeaver), 25
EvoWeb, 35, 65, 66, 68, 69
ExampleStreptomycesData, 26, 36
ExoLabel, 20, 21, 36
ExpandDiagonal, 11, 40
ExtractBy, 41
FastQFromSRR, 43
FindSets, 14, 44
FindSynteny, 5, 7, 11, 14, 22, 24, 41, 42, 54, 56, 70, 72, 80, 81
Fitch parsimony, 45
GainLoss.EvoWeaver (EvoWeaver-PPPreds), 29
Generic, 47
gffToDataFrame, 48
glm, 8
Hamming.EvoWeaver (EvoWeaver-PPPreds), 29
Jaccard-Robinson-Foulds Distance, 32, 57, 58
Jaccard.EvoWeaver (EvoWeaver-PPPreds), 29
JRFDist (PhyloDistance-JRFDist), 60
KFDist (PhyloDistance-KFDist), 62
Kuhner-Felsenstein Distance, 32, 58
lapply, 13
LinkedPairs, 49
LinkedPairs-class (LinkedPairs), 49
list, 52
MakeBlastDb, 3, 4, 50
MirrorTree.EvoWeaver (EvoWeaver-PSPreds), 31
Moran’s 1, 68
MoransI, 28, 51
MutualInformation.EvoWeaver (EvoWeaver-PPPreds), 29
NucleotideOverlap, 5, 11, 41, 53, 56, 70, 72, 79, 81
NVD.T.EvoWeaver (EvoWeaver-SLPreds), 33
Nye Similarity, 32
PairSummaries, 5, 7, 14, 41, 42, 44, 54, 72, 79
palette, 65
PhyloDistance, 32, 33, 57, 59, 60, 62, 63
PhyloDistance-CI (PhyloDistance-CIDist), 59
PhyloDistance-CIDist, 59
PhyloDistance-JRF (PhyloDistance-JRFDist), 60
PhyloDistance-JRFDist, 60
PhyloDistance-KF (PhyloDistance-KFDist), 62
PhyloDistance-KFDist, 62
PhyloDistance-RF (PhyloDistance-RFDist), 63
PhyloDistance-RFDist, 63
plot.EvoWeb, 35, 64
predict.EvoWeaver, 25, 26, 29, 31, 33–35, 64, 65, 66
PrepareSeqs, 69, 81
print.LinkedPairs (LinkedPairs), 49
print.simMat (simMat), 74
ProfDCA.EvoWeaver (EvoWeaver-PPPreds), 29
rapply, 12, 13
read.table, 37
ResidueMI.EvoWeaver (EvoWeaver-SLPreds), 33
RFDist (PhyloDistance-RFDist), 63
Robinson-Foulds Distance, 32, 57, 58
SelectByK, 71
Sequence Similarity, 73
simMat, 35, 74
simMat-class (simMat), 74
SpeciesTree (EvoWeaver), 25
subset, 77
subset.dendrogram, 77
SubSetPairs, 78
SummarizePairs, 11, 70, 79
SuperTree, 26, 28, 30, 81, 83
SuperTreeEx, 82, 83
Synteny, 22, 24

tempdir, 38
TMPDIR, 50
TranscripMI.EvoWeaver
   (EvoWeaver-GOPreds), 28
TreeDistance.EvoWeaver
   (EvoWeaver-PSPreds), 31
TreeLine, 82
XStringSet, 3, 50