Package 'TreeDist'

September 17, 2020

Type Package

Title Distances Between Phylogenetic Trees

Version 1.2.1

License GPL (>= 3)

Description Implements measures of tree similarity, including
information-based generalized Robinson-Foulds distances
(Phylogenetic Information Distance, Clustering Information Distance,
Matching Split Information Distance; Smith, 2020)
<doi:10.1093/bioinformatics/btaa614>;
Jaccard-Robinson-Foulds distances (Bocker et al. 2013)
<doi:10.1007/978-3-642-40453-5_13>,
including the Nye et al. (2006) metric <doi:10.1093/bioinformatics/bti720>;
the Matching Split Distance (Bogdanowicz & Giaro 2012)
<doi:10.1109/TCBB.2011.48>;
Maximum Agreement Subtree distances;
the Kendall-Colijn (2016) distance <doi:10.1093/molbev/msw124>, and the
Nearest Neighbour Interchange (NNI) distance, approximated per Li et al.
(1996) <doi:10.1007/3-540-61332-3_168>.
Calculates the median of a set of trees under any distance metric.

Copyright Incorporates Jonker-Volgenant Linear Assignment Problem
implementation by Roy Jonker, modified by Yong Yang after Yi
Cao.

URL https://ms609.github.io/TreeDist/,
https://github.com/ms609/TreeDist/

BugReports https://github.com/ms609/TreeDist/issues/

Depends R (>= 3.4.0), stats

Imports ape (>= 5.0), colorspace, memoise, phangorn (>= 2.2.1),
Rdpack, TreeTools (>= 1.1.0)

Suggests bookdown, cluster, kdensity, knitr, MASS, Quartet, rmarkdown,
Rcpp, testthat, Ternary (>= 1.1.2), TreeDistData (> 0.1.0),
TreeSearch, vdiffr

Additional_repositories https://ms609.github.io/packages/

RdMacros Rdpack

VignetteBuilder knitr

LinkingTo Rcpp

SystemRequirements C++11

LazyData true

ByteCompile true

Encoding UTF-8

Language en-GB

X-schema.org-keywords phylogenetics, tree-distance

RoxygenNote 7.1.1

NeedsCompilation yes

Author Martin R. Smith [aut, cre, cph, prg]
(<https://orcid.org/0000-0001-5660-1727>),
Roy Jonker [prg, cph],
Yong Yang [ctb, cph],
Yi Cao [ctb, cph]

Maintainer Martin R. Smith <martin.smith@durham.ac.uk>

Repository CRAN

Date/Publication 2020-09-17 13:10:15 UTC

R topics documented:

`AllSplitPairings` *Variation of information for all split pairings*

Description

Calculate the variation of clustering information (Meila 2007) for each possible pairing of non-trivial splits on *n* leaves, tabulating the number of pairings with each similarity.

Usage

```
AllSplitPairings(n)
```

Arguments

`n`	Integer specifying the number of leaves in a tree.

Value

`AllSplitPairings()` returns a named vector. The name of each element corresponds to a certain variation of information, in bits; the value of each element specifies the number of pairings of non-trivial splits that give rise to that variation of information. Split `AB|CD` is treated as distinct from `CD|AB`. If pairing `AB|CD=CD|AB` is considered equivalent to `CD|AB=CD|AB` (etc), then values should be divided by four.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

MeilÄ*f* M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.

Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing phylogenetic trees." *Bioinformatics*, online ahead of print. doi: 10.1093/bioinformatics/btaa614.

Examples

```
AllSplitPairings(6)
# Treat equivalent splits as identical by dividing by four:
AllSplitPairings(6) / 4L
```

`ClusteringEntropy` *Clustering entropy of all splits within a tree*

Description

Sum the entropy (`ClusteringEntropy()`) or information content (`ClusteringInfo()`) across each split within a phylogenetic tree, treating each split as dividing the leaves of the tree into two clusters (*sensu* Meila 2007; Vinh *et al.* 2010).

Usage

```
ClusteringEntropy(x)

ClusteringInfo(x)

## S3 method for class 'phylo'
ClusteringEntropy(x)

## S3 method for class 'list'
ClusteringEntropy(x)

## S3 method for class 'multiPhylo'
ClusteringEntropy(x)

## S3 method for class 'Splits'
ClusteringEntropy(x)

## S3 method for class 'phylo'
ClusteringInfo(x)

## S3 method for class 'list'
ClusteringInfo(x)

## S3 method for class 'multiPhylo'
ClusteringInfo(x)

## S3 method for class 'Splits'
ClusteringInfo(x)
```

Arguments

`x`	A tree of class `phylo`, a list of trees, or a `multiPhylo` object.

Details

Clustering entropy addresses the question "how much information is contained in the splits within a tree". Its approach is complementary to the phylogenetic information content, used in `SplitwiseInfo()`.

In essence, it asks, given a split that subdivides the leaves of a tree into two partitions, how easy it is to predict which partition a randomly drawn leaf belongs to.

Formally, the entropy of a split *S* that divides *n* leaves into two partitions of sizes *a* and *b* is given by $H(S) = - a/n \log a/n - b/n \log b/n$.

Base 2 logarithms are conventionally used, such that entropy is measured in bits. Entropy denotes the number of bits that are necessary to encode the outcome of a random variable: here, the random variable is "what partition does a randomly selected leaf belong to".

An even split has an entropy of 1 bit: there is no better way of encoding an outcome than using one bit to specify which of the two partitions the randomly selected leaf belongs to.

An uneven split has a lower entropy: membership of the larger partition is common, and thus less surprising; it can be signified using fewer bits in an optimal compression system.

If this sounds confusing, let's consider creating a code to transmit the cluster label of two randomly selected leaves. One straightforward option would be to use

- `00` = 'Both leaves belong to partition A'
- `11` = 'Both leaves belong to partition B'
- `01` = 'First leaf in A, second in B'
- `10` = 'First leaf in B, second in A'

This code uses two bits to transmit the partition labels of two leaves. If partitions A and B are equiprobable, this is the optimal code; our entropy – the average information content required per leaf – is 1 bit.

Alternatively, we could use the (suboptimal) code

- `0` = 'Both leaves belong to partition A'
- `111` = 'Both leaves belong to partition B'
- `101` = 'First leaf in A, second in B'
- `110` = 'First leaf in B, second in A'

If A is much larger than B, then most pairs of leaves will require just a single bit (code `0`). The additional bits when 1+ leaf belongs to B may be required sufficiently rarely that the average message requires fewer than two bits for two leaves, so the entropy is less than 1 bit. (The optimal coding strategy will depend on the exact sizes of A and B.)

As entropy measures the bits required to transmit the cluster label of each leaf (Vinh *et al.* 2010: p. 2840), the information content of a split is its entropy multiplied by the number of leaves.

Value

Returns the sum of the entropies or (clustering) information content, in bits, of each split in `x`.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

- MeilÄf M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.
- Vinh NX, Epps J, Bailey J (2010). "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance." *Journal of Machine Learning Research*, **11**, 2837–2854. doi: 10.1145/1553374.1553511.

See Also

Other information functions: `SplitEntropy()`, `SplitSharedInformation()`, `SplitwiseInfo()`

Examples

```
# Clustering entropy of an even split = 1 bit
ClusteringEntropy(TreeTools::as.Splits(c(rep(TRUE, 4), rep(FALSE, 4))))

# Clustering entropy of an uneven split < 1 bit
ClusteringEntropy(TreeTools::as.Splits(c(rep(TRUE, 2), rep(FALSE, 6))))

tree1 <- TreeTools::BalancedTree(8)
tree2 <- TreeTools::PectinateTree(8)

ClusteringInfo(tree1)
ClusteringEntropy(tree1)
ClusteringInfo(list(one = tree1, two = tree2))

ClusteringInfo(tree1) + ClusteringInfo(tree2)
ClusteringEntropy(tree1) + ClusteringEntropy(tree2)
ClusteringInfoDistance(tree1, tree2)
MutualClusteringInfo(tree1, tree2)
```

CompareAll *Distances between each pair of trees*

Description

Calculate the distance between each tree in a list, and each other tree in the same list.

Usage

```
CompareAll(x, Func, FUN.VALUE = Func(x[[1]], x[[1]], ...), ...)
```

Arguments

x	List of trees, in the format expected by `Func()`.
Func	distance function returning distance between two trees, e.g. `path.dist()`.
FUN.VALUE	Format of output of `Func()`, to be passed to `vapply()`. If unspecified, calculated by running `Func(x[[1]],x[[1]])`.
...	Additional parameters to pass to `Func()`.

Details

`CompareAll()` is not limited to tree comparisons: `Func` can be any symmetric function.

Value

`CompareAll()` returns a distance matrix of class `dist` detailing the distance between each pair of trees. Identical trees are assumed to have zero distance.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

```
# Generate a list of trees to compare
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)
trees <- list(bal1 = BalancedTree(1:8),
              pec1 = PectinateTree(1:8),
              pec2 = PectinateTree(c(4:1, 5:8)))

# Compare each tree with each other tree
CompareAll(trees, NNIDist)

# Providing FUN.VALUE yields a modest speed gain:
dist <- CompareAll(trees, NNIDist, FUN.VALUE = integer(7))

# View distances as a matrix
as.matrix(dist$lower)
```

`Entropy` *Entropy in bits*

Description

Calculate the entropy of a vector of probabilities, in bits. Probabilities should sum to one. Probabilities equalling zero will be ignored.

Usage

```
Entropy(...)
```

Arguments

`...`	Numerics or numeric vector specifying probabilities of outcomes.

Value

`Entropy()` returns the entropy of the specified probabilities, in bits.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

```
Entropy(1/2, 0, 1/2) # = 1
Entropy(rep(1/4, 4)) # = 2
```

`JaccardRobinsonFoulds` *Jaccard-Robinson-Foulds metric*

Description

Calculate the Jaccard-Robinson-Foulds metric (Böcker *et al.* 2013), a Generalized Robinson-Foulds metric.

Usage

```
JaccardRobinsonFoulds(
  tree1,
  tree2 = tree1,
  k = 1L,
  allowConflict = TRUE,
  similarity = FALSE,
  normalize = FALSE,
  reportMatching = FALSE
)

JaccardSplitSimilarity(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  k = 1L,
  allowConflict = TRUE,
  reportMatching = FALSE
)
```

Arguments

`tree1`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
`tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
`k`	An arbitrary exponent to which to raise the Jaccard index. Integer values greater than one are anticipated by Böcker *et al.* The Nye *et al.* metric uses `k = 1`. As `k` increases towards infinity, the metric converges to the Robinson-Foulds metric.

allowConflict	Logical specifying whether to allow conflicting splits to be paired. If FALSE, such pairings will be allocated a similarity score of zero.
similarity	Logical specifying whether to report the result as a tree similarity, rather than a difference.
normalize	If a numeric value is provided, this will be used as a maximum value against which to rescale results. If TRUE, results will be rescaled against a maximum value calculated from the specified tree sizes and topology, as specified in the 'Normalization' section below. If FALSE, results will not be rescaled.
reportMatching	Logical specifying whether to return the clade matchings as an attribute of the score.
splits1	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
splits2	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
nTip	(Optional) Integer specifying the number of leaves in each split.

Details

In short, the Jaccard-Robinson-Foulds metric is a generalized Robinson-Foulds metric: it finds the optimal matching that pairs each split in one tree with a similar split in the second. Matchings are scored according to the size of the largest split that is consistent with both of them, normalized against the Jaccard index, and raised to an arbitrary exponent. A more detailed explanation is provided in the vignettes.

By default, conflicting splits may be paired.

Note that the settings k = 1, allowConflict = TRUE, similarity = TRUE give the similarity metric of Nye *et al.* (2006); a slightly faster implementation of this metric is available as `NyeSimilarity()`.

The examples section below details how to visualize matchings with non-default parameter values.

Value

`JaccardRobinsonFoulds()` returns an array of numerics providing the distances between each pair of trees in `tree1` and `tree2`, or `splits1` and `splits2`.

Normalization

If `normalize = TRUE`, then results will be rescaled from zero to one by dividing by the maximum possible value for trees of the given topologies, which is equal to the sum of the number of splits in each tree. You may wish to normalize instead against the maximum number of splits present in a pair of trees with n leaves, by specifying `normalize = n - 3`.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

- Nye TMW, LiÃ² P, Gilks WR (2006). "A novel algorithm and web-based tool for comparing two alternative phylogenetic trees." *Bioinformatics*, **22**(1), 117–119. doi: 10.1093/bioinformatics/bti720.
- BÃ¶cker S, Canzar S, Klau GW (2013). "The generalized Robinson-Foulds metric." In Darling A, Stoye J (eds.), *Algorithms in Bioinformatics. WABI 2013. Lecture Notes in Computer Science, vol 8126*, 156–169. Springer, Berlin, Heidelberg. doi: 10.1007/9783642404535_13.

See Also

Other tree distances: `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
set.seed(2)
tree1 <- ape::rtree(10)
tree2 <- ape::rtree(10)
JaccardRobinsonFoulds(tree1, tree2, k = 2, allowConflict = FALSE)
JaccardRobinsonFoulds(tree1, tree2, k = 2, allowConflict = TRUE)

JRF2 <- function (tree1, tree2, ...)
  JaccardRobinsonFoulds(tree1, tree2, k = 2, allowConflict = FALSE, ...)

VisualizeMatching(JRF2, tree1, tree2, matchZeros = FALSE)
```

`KendallColijn` *Kendall-Colijn distance*

Description

Calculate the Kendall-Colijn tree distance, a measure related to the path difference.

Usage

```
KendallColijn(tree1, tree2 = tree1)

KCVector(tree)
```

Arguments

`tree1, tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
`tree`	A tree of class `phylo`.

Details

The Kendall-Colijn distance works by measuring, for each pair of leaves, the distance from the most recent common ancestor of those leaves and the root node. For a given tree, this produces a vector of values recording the distance-from-the-root of each most recent common ancestor of each pair of leaves.

Two trees are compared by taking the Euclidian distance between the respective vectors. This is calculated by taking the square root of the sum of the squares of the differences between the vectors.

This metric emphasizes the position of the root; the path difference instead measures the distance of the last common ancestor of each pair of leaves from the leaves themselves, i.e. the length of the path from one leaf to another.

Value

`KendallColijn()` returns an array of numerics providing the distances between each pair of trees in `tree1` and `tree2`, or `splits1` and `splits2`.

Functions

- `KCVector`: Creates a vector that characterises a rooted tree, as described in Kendall & Colijn (2016).

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Kendall M, Colijn C (2016). "Mapping phylogenetic trees to reveal distinct patterns of evolution." *Molecular Biology and Evolution*, **33**(10), 2735–2743. doi: 10.1093/molbev/msw124.

See Also

`treespace::treeDist` is a more sophisticated, if more cumbersome, implementation that supports lambda > 0, i.e. use of edge lengths in tree comparison.

Other tree distances: `JaccardRobinsonFoulds()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
KendallColijn(TreeTools::BalancedTree(8), TreeTools::PectinateTree(8))

set.seed(0)
KendallColijn(TreeTools::BalancedTree(8), lapply(rep(8, 3), ape::rtree))
KendallColijn(lapply(rep(8, 4), ape::rtree))
```

LAPJV *Solve linear assignment problem using LAPJV*

Description

Use the algorithm of Jonker & Volgenant (1987) to solve the Linear Sum Assignment Problem.

Usage

```
LAPJV(x)
```

Arguments

x	Square matrix of costs.

Details

The Linear Assignment Problem seeks to match each row of a matrix with a column, such that the cost of the matching is minimized.

The Jonker & Volgenant approach is a faster alternative to the Hungarian algorithm (Munkres 1957), which is implemented in `clue::solve_LSAP()`.

Note: the JV algorithm expects integers. In order to apply the function to a non-integer n, as in the tree distance calculations in this package, each n is multiplied by the largest available integer before applying the JV algorithm. If two values of n exhibit a trivial difference – e.g. due to floating point errors – then this can lead to interminable run times. (If numbers of the magnitude of billions differ only in their last significant digit, then the JV algorithm may undergo billions of iterations.) To avoid this, integers over 2^22 that differ by a value of 8 or less are treated as equal.

NB. At present, only square matrices are supported; if you need support for non-square matrices, drop a note at issue #25 and I'll prioritize development.

Value

A list with two entries: `score`, the score of the optimal matching; and `matching`, the columns matched to each row of the matrix in turn.

Author(s)

C++ code by Roy Jonker, MagicLogic Optimization Inc. roy_jonker@magiclogic.com, with contributions from Yong Yang yongyanglink@gmail.com, after Yi Cao

References

Jonker R, Volgenant A (1987). "A shortest augmenting path algorithm for dense and sparse linear assignment problems." *Computing*, **38**, 325–340. doi: 10.1007/BF02278710.

Munkres J (1957). "Algorithms for the assignment and transportation problems." *Journal of the Society for Industrial and Applied Mathematics*, **5**(1), 32–38. doi: 10.1137/0105003.

Examples

```
problem <- matrix(c(7, 9, 8, 9,
                    2, 8, 5, 7,
                    1, 6, 6, 9,
                    3, 6, 2, 2), 4, 4, byrow=TRUE)

LAPJV(problem)
```

MASTSize *Maximum Agreement Subtree size*

Description

Calculate the size or phylogenetic information content (Steel & Penny 2006) of the maximum agreement subtree between two phylogenetic trees, i.e. the largest tree that can be obtained from both `tree1` and `tree2` by deleting, but not rearranging, leaves, using the algorithm of Valiente (2009).

Usage

```
MASTSize(tree1, tree2 = tree1, rooted = TRUE)

MASTInfo(tree1, tree2 = tree1, rooted = TRUE)
```

Arguments

`tree1, tree2`	Trees of class `phylo`, or lists of such trees to undergo pairwise comparison.
`rooted`	Logical specifying whether to treat the trees as rooted.

Details

Implemented for trees with up to 4096 tips. Contact the maintainer if you need to process larger trees.

Value

`MASTSize()` returns an integer specifying the number of leaves in the maximum agreement subtree.

`MASTInfo()` returns a vector or matrix listing the phylogenetic information content, in bits, of the maximum agreement subtree.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multistate characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

Valiente G (2009). *Combinatorial Pattern Matching Algorithms in Computational Biology using Perl and R*, CRC Mathematical and Computing Biology Series. CRC Press, Boca Raton.

See Also

`phangorn::mast()`, a slower implementation that also lists the leaves contained within the subtree.

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MatchingSplitDistance()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
 # for as.phylo, BalancedTree, PectinateTree:
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)

MASTSize(PectinateTree(8), BalancedTree(8))
MASTInfo(PectinateTree(8), BalancedTree(8))

MASTSize(BalancedTree(7), as.phylo(0:3, 7))
MASTSize(as.phylo(0:3, 7), PectinateTree(7))

MASTInfo(BalancedTree(7), as.phylo(0:3, 7))
MASTInfo(as.phylo(0:3, 7), PectinateTree(7))

MASTSize(list(Bal = BalancedTree(7), Pec = PectinateTree(7)),
         as.phylo(0:3, 7))
MASTInfo(list(Bal = BalancedTree(7), Pec = PectinateTree(7)),
         as.phylo(0:3, 7))

CompareAll(as.phylo(0:4, 8), MASTSize)
CompareAll(as.phylo(0:4, 8), MASTInfo)
```

MatchingSplitDistance *Matching Split Distance*

Description

Calculate the Matching Split Distance (Bogdanowicz and Giaro 2012; Lin *et al.* 2012) for unrooted binary trees.

Usage

```
MatchingSplitDistance(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE
)

MatchingSplitDistanceSplits(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  normalize = TRUE,
  reportMatching = FALSE
)
```

Arguments

tree1	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
normalize	If a numeric value is provided, this will be used as a maximum value against which to rescale results. If TRUE, results will be rescaled against a maximum value calculated from the specified tree sizes and topology, as specified in the 'Normalization' section below. If FALSE, results will not be rescaled.
reportMatching	Logical specifying whether to return the clade matchings as an attribute of the score.
splits1	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
splits2	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
nTip	(Optional) Integer specifying the number of leaves in each split.

Value

MatchingSplitDistance() returns an array of numerics providing the distances between each pair of trees in tree1 and tree2, or splits1 and splits2.

Normalization

A normalization value or function must be provided in order to return a normalized value. If you are aware of a generalised formula, please let me know by creating a GitHub issue so that it can be implemented.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Bogdanowicz D, Giaro K (2012). "Matching split distance for unrooted binary phylogenetic trees." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **9**(1), 150–160. doi: 10.1109/TCBB.2011.48.

Lin Y, Rajan V, Moret BME (2012). "A metric for phylogenetic trees based on matching." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **4**(9), 1014–1022. doi: 10.1109/TCBB.2011.157.

See Also

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
MatchingSplitDistance(lapply(rep(8, 5), ape::rtree), normalize = 16)

MatchingSplitDistance(TreeTools::BalancedTree(6),
                      TreeTools::PectinateTree(6),
                      reportMatching = TRUE)

VisualizeMatching(MatchingSplitDistance, TreeTools::BalancedTree(6),
                  TreeTools::PectinateTree(6))
```

---

median.multiPhylo *Median of a set of trees*

Description

Calculate the single binary tree that represents the geometric median – an 'average' – of a forest of tree topologies.

Usage

```
## S3 method for class 'multiPhylo'
median(
  x,
  na.rm = FALSE,
  Distance = ClusteringInfoDistance,
  index = FALSE,
  breakTies = TRUE,
  ...
)
```

Arguments

x	Object of class multiPhylo containing phylogenetic trees.
na.rm, ...	Unused; included for consistency with default function..
Distance	Function to calculate distances between each pair of trees in x.
index	Logical: if TRUE, return the index of the median tree(s); if FALSE, return the tree itself.
breakTies	Logical: if TRUE, return a single tree with the minimum score; if FALSE, return all tied trees.

Details

The geometric median is the tree that exhibits the shortest average distance from each other tree topology in the set. It represents an 'average' of a set of trees, though note that an unsampled tree may be closer to the geometric 'centre of gravity' of the input set – such a tree would not be considered.

The result will depend on the metric chosen to calculate distances between tree topologies. In the absence of a natural metric of tree topologies, the default choice is ClusteringInfoDistance() – which discards branch length information. If specifying a different function, be sure that it returns a difference, rather than a similarity.

Value

median() returns an object of class phylo corresponding to the geometric median of a set of trees: that is, the tree whose average distance from all other trees in the set is lowest. If multiple trees tie in their average distance, the first will be returned, unless breakTies = FALSE, in which case an object of class multiPhylo containing all such trees will be returned.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Consensus methods: ape::consensus(), TreeTools::ConsensusWithout()

Examples

```
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)
tenTrees <- as.phylo(1:10, nTip = 8)

# Default settings:
median(tenTrees)

# Robinson-Foulds distances include ties:
median(tenTrees, Distance = RobinsonFoulds, breakTies = FALSE)

# Be sure to use a distance function, rather than a similarity:
NyeDistance <- function (...) NyeSimilarity(..., similarity = FALSE)
```

```
median(tenTrees, Distance = NyeDistance)

# To analyse a list of trees that is not of class multiPhylo:
treeList <- lapply(1:10, as.phylo, nTip = 8)
class(treeList)
median(structure(treeList, class = 'multiPhylo'))
```

`MeilaVariationOfInformation` *Use variation of clustering information to compare pairs of splits*

Description

Compare a pair of splits viewed as clusterings of taxa, using the variation of clustering information proposed by Meila (2007).

Usage

```
MeilaVariationOfInformation(split1, split2)

MeilaMutualInformation(split1, split2)
```

Arguments

`split1, split2`	Logical vectors listing leaves in a consistent order, identifying each leaf as a member of the ingroup (`TRUE`) or outgroup (`FALSE`) of the split in question.

Details

This is equivalent to the mutual clustering information (Vinh *et al.* 2010). For the total information content, multiply the VoI by the number of leaves.

Value

`MeilaVariationOfInformation()` returns the variation of (clustering) information, measured in bits.

`MeilaMutualInformation()` returns the mutual information, measured in bits.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

MeilÄ*f* M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.

Vinh NX, Epps J, Bailey J (2010). "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance." *Journal of Machine Learning Research*, **11**, 2837–2854. doi: 10.1145/1553374.1553511.

Examples

```
# Maximum variation = information content of each split separately
A <- TRUE
B <- FALSE
MeilaVariationOfInformation(c(A, A, A, B, B, B), c(A, A, A, A, A, A))
Entropy(c(3, 3) / 6) + Entropy(c(0, 6) / 6)

# Minimum variation = 0
MeilaVariationOfInformation(c(A, A, A, B, B, B), c(A, A, A, B, B, B))

# Not always possible for two evenly-sized splits to reach maximum
# variation of information
Entropy(c(3, 3) / 6) * 2  # = 2
MeilaVariationOfInformation(c(A, A, A,B ,B, B), c(A, B, A, B, A, B)) # < 2

# Phylogenetically uninformative groupings contain spliting information
Entropy(c(1, 5) / 6)
MeilaVariationOfInformation(c(B, A, A, A, A, A), c(A, A, A, A, A, B))
```

NNIDist *Approximate Nearest Neighbour Interchange distance*

Description

Use the approach of Li *et al.* (1996) to approximate the Nearest Neighbour Interchange distance (Robinson, 1971) between phylogenetic trees.

Usage

```
NNIDist(tree1, tree2 = tree1)

NNIDiameter(tree)
```

Arguments

`tree1, tree2`	Single trees of class `phylo` to undergo comparison.
`tree`	Object of supported class representing a tree or list of trees, or an integer specifying the number of leaves in a tree/trees.

Details

In brief, this approximation algorithm works by identifying edges in one tree that do not match edges in the second. Each of these edges must undergo at least one NNI operation in order to reconcile the trees. Edges that match in both trees need never undergo an NNI operation, and divide each tree into smaller regions. By 'cutting' matched edges into two, a tree can be divided into a number of regions that solely comprise unmatched edges.

These regions can be viewed as separate trees that need to be reconciled. One way to reconcile these trees is to conduct a series of NNI operations that reduce a tree to a pectinate (caterpillar) tree,

then to conduct an analogue of the mergesort algorithm. This takes at most $n \log n + O(n)$ NNI operations, and provides a loose upper bound on the NNI score. The maximum number of moves for an n-leaf tree (OEIS A182136) can be calculated exactly for small trees (Fack *et al.* 2002); this provides a tighter upper bound, but is unavailable for $n > 12$. `NNIDiameter()` reports the limits on this bound.

Leaves:	1	2	3	4	5	6	7	8	9	10	11	12	13
Diameter:	0	0	0	1	3	5	7	10	12	15	18	21	?

Value

`NNIDist()` returns, for each pair of trees, a named vector containing three integers:

- `lower` is a lower bound on the NNI distance, and corresponds to the RF distance between the trees.
- `tight_upper` is an upper bound on the distance, based on calculated maximum diameters for trees with < 13 leaves. *NA* is returned if trees are too different to employ this approach.
- `loose_upper` is a looser upper bound on the distance, using $n \log n + O(n)$.

`NNIDiameter()` returns a matrix specifying (bounds on) the diameter of the NNI distance metric on the specified tree(s). Columns correspond to:

- `liMin`:
$$n - 3$$
, a lower bound on the diameter (Li *et al.* 1996);
- `fackMin`: Lower bound on diameter following Fack *et al.* (2002), i.e.
$$\log 2N!/4$$
;
- `min`: The larger of `liMin` and `fackMin`;
- `exact`: The exact value of the diameter, where $n < 13$;
- `liMax`: Upper bound on diameter following Li *et al.* (1996), i.e.
$$n \log 2n + O(n)$$
;
- `fackMax`: Upper bound on diameter following Fack *et al.* (2002), i.e. (
$$N - 2$$
) ceiling(
$$\log 2n$$
)
  - N;
- `max`: The smaller of `liMax` and `fackMax`;

where n is the number of leaves, and N the number of internal nodes, i.e.

$$n - 2$$

.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Fack V, Lievens S, Van der Jeugt J (2002). "On the diameter of the rotation graph of binary coupling trees." *Discrete Mathematics*, **245**(1-3), 1–18. doi: 10.1016/S0012365X(01)004186.

Li M, Tromp J, Zhang L (1996). "Some notes on the nearest neighbour interchange distance." In Goos G, Hartmanis J, Leeuwen J, Cai J, Wong CK (eds.), *Computing and Combinatorics*, volume 1090, 343–351. Springer, Berlin, Heidelberg. ISBN 978-3-540-61332-9 978-3-540-68461-9, doi: 10.1007/3540613323_168.

Robinson D (1971). "Comparison of labeled trees with valency three." *Journal of Combinatorial Theory, Series B*, **11**(2), 105–119. doi: 10.1016/00958956(71)900207.

See Also

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)

NNIDist(BalancedTree(7), PectinateTree(7))

NNIDist(BalancedTree(7), as.phylo(0:2, 7))
NNIDist(as.phylo(0:2, 7), PectinateTree(7))

NNIDist(list(bal = BalancedTree(7), pec = PectinateTree(7)),
        as.phylo(0:2, 7))

CompareAll(as.phylo(30:33, 8), NNIDist)
```

`NyeSimilarity` *Nye* et al. *(2006) tree comparison*

Description

`NyeSimilarity()` and `NyeSplitSimilarity()` implement the Generalized Robinson-Foulds tree comparison metric of Nye *et al.* (2006). In short, this finds the optimal matching that pairs each branch from one tree with a branch in the second, where matchings are scored according to the size of the largest split that is consistent with both of them, normalized against the Jaccard index. A more detailed account is available in the vignettes.

Usage

```
NyeSimilarity(
  tree1,
  tree2 = tree1,
  similarity = TRUE,
  normalize = FALSE,
  normalizeMax = !is.logical(normalize),
  reportMatching = FALSE,
  diag = TRUE
)

NyeSplitSimilarity(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)
```

Arguments

tree1	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
similarity	Logical specifying whether to report the result as a tree similarity, rather than a difference.
normalize	If a numeric value is provided, this will be used as a maximum value against which to rescale results. If TRUE, results will be rescaled against a maximum value calculated from the specified tree sizes and topology, as specified in the 'Normalization' section below. If FALSE, results will not be rescaled.
normalizeMax	When calculating similarity, normalize against the maximum number of splits that could have been present (TRUE), or the number of splits that were actually observed (FALSE)? Defaults to the number of splits in the better-resolved tree; set normalize = pmin.int to use the number of splits in the less resolved tree.
reportMatching	Logical specifying whether to return the clade matchings as an attribute of the score.
diag	Logical specifying whether to return similarities along the diagonal, i.e. of each tree with itself. Applies only if tree2 is a list identical to tree1, or NULL.
splits1	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
splits2	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
nTip	(Optional) Integer specifying the number of leaves in each split.

Details

The measure is defined as a similarity score. If `similarity = FALSE`, the similarity score will be converted into a distance by doubling it and subtracting it from the number of splits present in both trees. This ensures consistency with `JaccardRobinsonFoulds`.

Note that `NyeSimilarity(tree1, tree2)` is equivalent to, but slightly faster than, `JaccardRobinsonFoulds(tree1, tree2, k = 1, allowConflict = TRUE)`.

Value

`NyeSimilarity()` returns an array of numerics providing the distances between each pair of trees in `tree1` and `tree2`, or `splits1` and `splits2`.

Normalization

If `normalize = TRUE` and `similarity = TRUE`, then results will be rescaled from zero to one by dividing by the mean number of splits in the two trees being compared.

You may wish to normalize instead against the number of splits present in the smaller tree, which represents the maximum value possible for a pair of trees with the specified topologies (`normalize = pmin.int`); the number of splits in the most resolved tree (`normalize = pmax.int`); or the maximum value possible for any pair of trees with *n* leaves, *n* - 3 (`normalize = TreeTools::NTip(tree1) - 3L`).

If `normalize = TRUE` and `similarity = FALSE`, then results will be rescaled from zero to one by dividing by the total number of splits in the pair of trees being considered.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Nye TMW, LiÃ² P, Gilks WR (2006). "A novel algorithm and web-based tool for comparing two alternative phylogenetic trees." *Bioinformatics*, **22**(1), 117–119. doi: 10.1093/bioinformatics/bti720.

See Also

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TreeDistance()`

Examples

```
library('TreeTools')
NyeSimilarity(BalancedTree(8), PectinateTree(8))
VisualizeMatching(NyeSimilarity ,BalancedTree(8), PectinateTree(8))
NyeSimilarity(as.phylo(0:5, nTip = 8), PectinateTree(8))
NyeSimilarity(as.phylo(0:5, nTip = 8), similarity = FALSE)
```

PathDist *Path distance*

Description

Calculate the path distance between trees.

Usage

```
PathDist(tree1, tree2 = NULL)
```

Arguments

tree1, tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.

Details

This function is a wrapper for the function path.dist() in the phangorn package. It pre-processes trees to ensure that their internal representation does not cause the path.dist() function to crash R.

The path distance is also termed the cladistic difference or topological distance.

Use of the path distance is discouraged as it emphasizes shallow relationships at the expense of deeper (and arguably more fundamental) relationships (Farris, 1973).

Value

PathDist() returns a vector or distance matrix of distances between trees.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Farris JS (1973). "On comparing the shapes of taxonomic trees." *Systematic Zoology*, **22**(1), 50–54. doi: 10.2307/2412378.

See Also

Other tree distances: JaccardRobinsonFoulds(), KendallColijn(), MASTSize(), MatchingSplitDistance(), NNIDist(), NyeSimilarity(), Robinson-Foulds, SPRDist(), TreeDistance()

Examples

```
library('TreeTools')

PathDist(BalancedTree(7), PectinateTree(7))

PathDist(BalancedTree(7), as.phylo(0:2, 7))
PathDist(as.phylo(0:2, 7), PectinateTree(7))

PathDist(list(bal = BalancedTree(7), pec = PectinateTree(7)),
        as.phylo(0:2, 7))

CompareAll(as.phylo(30:33, 8), PathDist)
```

Robinson-Foulds *Robinson-Foulds distances, with adjustments for phylogenetic information content*

Description

Calculate the Robinson-Foulds distance, or the equivalent similarity measure, with options to (i) annotate matched splits; (ii) weight splits according to their phylogenetic information content (Smith 2020).

Usage

```
InfoRobinsonFoulds(
  tree1,
  tree2 = tree1,
  similarity = FALSE,
  normalize = FALSE,
  reportMatching = FALSE
)

InfoRobinsonFouldsSplits(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)

RobinsonFoulds(
  tree1,
  tree2 = tree1,
  similarity = FALSE,
  normalize = FALSE,
  reportMatching = FALSE
)
```

```
RobinsonFouldsMatching(
  tree1,
  tree2 = tree1,
  similarity = FALSE,
  normalize = FALSE,
  ...
)

RobinsonFouldsSplits(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)
```

Arguments

tree1	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
similarity	Logical specifying whether to report the result as a tree similarity, rather than a difference.
normalize	If a numeric value is provided, this will be used as a maximum value against which to rescale results. If TRUE, results will be rescaled against a maximum value calculated from the specified tree sizes and topology, as specified in the 'Normalization' section below. If FALSE, results will not be rescaled.
reportMatching	Logical specifying whether to return the clade matchings as an attribute of the score.
splits1	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
splits2	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective split.
nTip	(Optional) Integer specifying the number of leaves in each split.
...	Not used.

Details

Note that if reportMatching = TRUE, the pairScores attribute returns a logical matrix specifying whether each pair of splits is identical.

InfoRobinsonFoulds() calculates the tree similarity or distance by summing the phylogenetic information content of all splits that are (or are not) identical in both trees. Consequently, splits that

are more likely to be identical by chance alone make a smaller contribution to overall tree distance, because their similarity is less remarkable.

Value

`RobinsonFoulds()` and `InfoRobinsonFoulds()` return an array of numerics providing the distances between each pair of trees in `tree1` and `tree2`, or `splits1` and `splits2`.

Functions

- `RobinsonFouldsMatching`: Matched splits, intended for use with `VisualizeMatching()`.

Normalization

- `RobinsonFoulds()` is normalized against the total number of splits that are present.
- `InfoRobinsonFoulds()` is normalized against the sum of the phylogenetic information of all splits in both trees, treated independently.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Robinson DF, Foulds LR (1981). "Comparison of phylogenetic trees." *Mathematical Biosciences*, **53**(1-2), 131–147. doi: 10.1016/00255564(81)900432.

Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multistate characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing phylogenetic trees." *Bioinformatics*, online ahead of print. doi: 10.1093/bioinformatics/btaa614.

See Also

Display paired splits: `VisualizeMatching()`

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `SPRDist()`, `TreeDistance()`

Examples

```
 # For BalancedTree, PectinateTree, as.phylo:
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)
balanced7 <- BalancedTree(7)
pectinate7 <- PectinateTree(7)
RobinsonFoulds(balanced7, pectinate7)
RobinsonFoulds(balanced7, pectinate7, normalize = TRUE)
VisualizeMatching(RobinsonFouldsMatching, balanced7, pectinate7)

InfoRobinsonFoulds(balanced7, pectinate7)
VisualizeMatching(InfoRobinsonFoulds, balanced7, pectinate7)
```

`SplitEntropy` *Entropy of two splits*

Description

Calculate the entropy, joint entropy, entropy distance and information content of two splits, treating each split as a division of n leaves into two groups. Further details are available in a vignette, MacKay (2003) and Meila (2007).

Usage

```
SplitEntropy(split1, split2 = split1)
```

Arguments

`split1, split2`	Logical vectors listing leaves in a consistent order, identifying each leaf as a member of the ingroup (`TRUE`) or outgroup (`FALSE`) of the split in question.

Value

A numeric vector listing, in bits:

- `H1` The entropy of split 1;
- `H2` The entropy of split 2;
- `H12` The joint entropy of both splits;
- `I` The mutual information of the splits;
- `Hd` The entropy distance (variation of information) of the splits.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

MacKay DJC (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge. https://www.inference.org.uk/itprnn/book.pdf.

MeilÄ*f* M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.

See Also

Other information functions: `ClusteringEntropy()`, `SplitSharedInformation()`, `SplitwiseInfo()`

Examples

```
A <- TRUE
B <- FALSE
SplitEntropy(c(A, A, A, B, B, B), c(A, A, B, B, B, B))
```

`SplitsCompatible` *Are splits compatible?*

Description

Determine whether splits are compatible (concave); i.e. they can both occur on a single tree.

Usage

```
SplitsCompatible(split1, split2)
```

Arguments

`split1, split2`	Logical vectors listing leaves in a consistent order, identifying each leaf as a member of the ingroup (`TRUE`) or outgroup (`FALSE`) of the split in question.

Value

`SplitsCompatible()` returns a logical specifying whether the splits provided are compatible with one another.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

```
A <- TRUE
B <- FALSE
SplitsCompatible(c(A, A, A, B, B, B),
                 c(A, A, B, B, B, B))
SplitsCompatible(c(A, A, A, B, B, B),
                 c(A, A, B, B, B, A))
```

`SplitSharedInformation` *Shared information content of two splits*

Description

Calculate the phylogenetic information shared, or not shared, between two splits. See the accompanying vignette for definitions.

Usage

```
SplitSharedInformation(n, A1, A2 = A1)

SplitDifferentInformation(n, A1, A2 = A1)

TreesConsistentWithTwoSplits(n, A1, A2 = A1)

LnTreesConsistentWithTwoSplits(n, A1, A2 = A1)
```

Arguments

`n`	Integer specifying the number of leaves
`A1, A2`	Integers specifying the number of taxa in *A1* and *A2*, once the splits have been arranged such that *A1* fully overlaps with *A2*.

Details

Split *S1* divides *n* leaves into two splits, *A1* and *B1*. Split *S2* divides the same leaves into the splits *A2* and *B2*.

Splits must be named such that *A1* fully overlaps with *A2*: that is to say, all taxa in *A1* are also in *A2*, or *vice versa*. Thus, all taxa in the smaller of *A1* and *A2* also occur in the larger.

Value

`TreesConsistentWithTwoSplits()` returns the number of unrooted bifurcating trees consistent with two splits.

`SplitSharedInformation()` returns the phylogenetic information that two splits have in common, in bits.

`SplitDifferentInformation()` returns the amount of phylogenetic information distinct to one of the two splits, in bits.

Functions

- `SplitDifferentInformation`: Different information between two splits.
- `TreesConsistentWithTwoSplits`: Number of trees consistent with two splits.
- `LnTreesConsistentWithTwoSplits`: Natural logarithm of `TreesConsistentWithTwoSplits`.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

MeilÄ*f* M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.

See Also

Other information functions: `ClusteringEntropy()`, `SplitEntropy()`, `SplitwiseInfo()`

Examples

```
# Eight leaves, labelled A to H.
# Split 1: ABCD|EFGH
# Split 2: ABC|DEFGH
# Let A1 = ABCD (four taxa), and A2 = ABC (three taxa).
# A1 and A2 overlap (both contain ABC).

TreesConsistentWithTwoSplits(n = 8, A1 = 4, A2 = 3)
SplitSharedInformation(n = 8, A1 = 4, A2 = 3)
SplitDifferentInformation(n = 8, A1 = 4, A2 = 3)

# If splits are identical, then their shared information is the same
# as the information of either split:
SplitSharedInformation(n = 8, A1 = 3, A2 = 3)
TreeTools::SplitInformation(3, 5)
```

SplitwiseInfo *Information content of splits within a tree*

Description

Sum the phylogenetic information content for all splits within a phylogenetic tree. This value will be greater than the total information content of the tree where a tree contains multiple splits, as these splits will contain mutual information.

Usage

```
SplitwiseInfo(x)
```

Arguments

`x`	A tree of class `phylo`, a list of trees, or a `multiPhylo` object.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

An introduction to the phylogenetic information content of a split is given in `SplitInformation()` and in a package vignette.

Other information functions: `ClusteringEntropy()`, `SplitEntropy()`, `SplitSharedInformation()`

Examples

```
SplitwiseInfo(TreeTools::PectinateTree(8))
```

SPRDist *Approximate Subtree Prune and Regraft distance*

Description

Approximate the Subtree Prune and Regraft (SPR) distance.

Usage

```
SPRDist(tree1, tree2 = NULL, symmetric = TRUE)
```

Arguments

tree1, tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
symmetric	Logical specifying whether to produce a better heuristic by calculating the minimum of SPRDist(t1,t2) and SPRDist(t2,t1), which are not guaranteed to be equal due to the heuristic nature of the approximation (see phangorn#97). Set to FALSE for the faster approximation, as implemented in 'phangorn'.

Details

SPRDist() is a wrapper for the function SPR.dist() in the phangorn package. It pre-processes trees to ensure that their internal representation does not cause the SPR.dist() function to crash R, and allows an improved (but slower) symmetric heuristic.

A memory leak is present in phangorn v2.5.5. To avoid a drain on system resources, install the latest version of phangorn with devtools::install_github('KlausVigo/phangorn').

Value

SPRDist() returns a vector or distance matrix of distances between trees.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree distances: JaccardRobinsonFoulds(), KendallColijn(), MASTSize(), MatchingSplitDistance(), NNIDist(), NyeSimilarity(), PathDist(), Robinson-Foulds, TreeDistance()

Examples

```
library('TreeTools', quietly = TRUE, warn.conflicts = FALSE)

SPRDist(BalancedTree(7), PectinateTree(7))

SPRDist(BalancedTree(7), as.phylo(0:2, 7))
SPRDist(as.phylo(0:2, 7), PectinateTree(7))

SPRDist(list(bal = BalancedTree(7), pec = PectinateTree(7)),
        as.phylo(0:2, 7))

CompareAll(as.phylo(30:33, 8), SPRDist)
```

TreeDistance *Information-based generalized Robinson-Foulds distances*

Description

Calculate tree similarity and distance measures based on the amount of phylogenetic or clustering information that two trees hold in common, as proposed in Smith (2020).

Usage

```
TreeDistance(tree1, tree2 = tree1)

SharedPhylogeneticInfo(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE,
  diag = TRUE
)

DifferentPhylogeneticInfo(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE
)

PhylogeneticInfoDistance(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE
)
```

```
ClusteringInfoDistance(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE
)

ExpectedVariation(tree1, tree2, samples = 10000)

MutualClusteringInfo(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE,
  diag = TRUE
)

SharedPhylogeneticInfoSplits(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)

MutualClusteringInfoSplits(
  splits1,
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)

MatchingSplitInfo(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE,
  diag = TRUE
)

MatchingSplitInfoDistance(
  tree1,
  tree2 = tree1,
  normalize = FALSE,
  reportMatching = FALSE
)

MatchingSplitInfoSplits(
  splits1,
```

```
  splits2,
  nTip = attr(splits1, "nTip"),
  reportMatching = FALSE
)
```

Arguments

`tree1, tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison.
`normalize`	If a numeric value is provided, this will be used as a maximum value against which to rescale results. If `TRUE`, results will be rescaled against a maximum value calculated from the specified tree sizes and topology, as specified in the 'Normalization' section below. If `FALSE`, results will not be rescaled.
`reportMatching`	Logical specifying whether to return the clade matchings as an attribute of the score.
`diag`	Logical specifying whether to return similarities along the diagonal, i.e. of each tree with itself. Applies only if `tree2` is a list identical to `tree1`, or `NULL`.
`samples`	Integer specifying how many samplings to obtain; accuracy of estimate increases with `sqrt(samples)`.
`splits1, splits2`	Logical matrices where each row corresponds to a leaf, either listed in the same order or bearing identical names (in any sequence), and each column corresponds to a split, such that each leaf is identified as a member of the ingroup (`TRUE`) or outgroup (`FALSE`) of the respective split.
`nTip`	(Optional) Integer specifying the number of leaves in each split.

Details

Generalized Robinson-Foulds distances calculate tree similarity by finding an optimal matching that the similarity between a split on one tree and its pair on a second, considering all possible ways to pair splits between trees (including leaving a split unpaired).

The methods implemented here use the concepts of entropy and information (MacKay 2003) to assign a similarity score between each pair of splits.

The returned tree similarity measures state the amount of information, in bits, that the splits in two trees hold in common when they are optimally matched, following Smith (2020). The complementary tree distance measures state how much information is different in the splits of two trees, under an optimal matching.

Value

If `reportMatching = FALSE`, the functions return a numeric vector specifying the requested similarities or differences.

If `reportMatching = TRUE`, the functions additionally return details of which clades are matched in the optimal matching, which can be viewed using `VisualizeMatching()`.

Concepts of information

The phylogenetic (Shannon) information content and entropy of a split are defined in a separate vignette.

Using the mutual (clustering) information (Meila 2007, Vinh *et al.* 2010) of two splits to quantify their similarity gives rise to the Mutual Clustering Information measure (`MutualClusteringInfo()`, `MutualClusteringInfoSplits()`); the entropy distance gives the Clustering Information Distance (`ClusteringInfoDistance()`). This approach is optimal in many regards, and is implemented with normalization in the convenience function `TreeDistance()`.

Using the amount of phylogenetic information common to two splits to measure their similarity gives rise to the Shared Phylogenetic Information similarity measure (`SharedPhylogeneticInfo()`, `SharedPhylogeneticInfoSplits()`). The amount of information distinct to each of a pair of splits provides the complementary Different Phylogenetic Information distance metric (`DifferentPhylogeneticInfo()`).

The Matching Split Information measure (`MatchingSplitInfo()`, `MatchingSplitInfoSplits()`) defines the similarity between a pair of splits as the phylogenetic information content of the most informative split that is consistent with both input splits; `MatchingSplitInfoDistance()` is the corresponding measure of tree difference. (More information here.)

Conversion to distances:

To convert similarity measures to distances, it is necessary to subtract the similarity score from a maximum value. In order to generate distance *metrics*, these functions subtract the similarity twice from the total information content (SPI, MSI) or entropy (MCI) of all the splits in both trees (Smith 2020).

Normalization:

If `normalize = TRUE`, then results will be rescaled such that distance ranges from zero to (in principle) one. The maximum **distance** is the sum of the information content or entropy of each split in each tree; the maximum **similarity** is half this value. (See Vinh *et al.* (2010, table 3) and Smith (2020) for alternative normalization possibilities.)

Note that a distance value of one (= similarity of zero) will seldom be achieved, as even the most different trees exhibit some similarity. It may thus be helpful to rescale the normalized value such that the *expected* distance between a random pair of trees equals one. This can be calculated with `ExpectedVariation()`; or see package 'TreeDistData' for a compilation of expected values under different metrics for trees with up to 200 leaves.

Alternatively, to scale against the information content or entropy of all splits in the most or least informative tree, use normalize = `pmax` or `pmin` respectively. To calculate the relative similarity against a reference tree that is known to be 'correct', use normalize = ``SplitwiseInfo(trueTree) (SPI, MSI) or `ClusteringEntropy(trueTree)` (MCI).

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

- MacKay DJC (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge. https://www.inference.org.uk/itprnn/book.pdf.

- MeilÄƒ M (2007). "Comparing clusterings—an information based distance." *Journal of Multivariate Analysis*, **98**(5), 873–895. doi: 10.1016/j.jmva.2006.11.013.
- Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing phylogenetic trees." *Bioinformatics*, online ahead of print. doi: 10.1093/bioinformatics/btaa614.
- Vinh NX, Epps J, Bailey J (2010). "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance." *Journal of Machine Learning Research*, **11**, 2837–2854. doi: 10.1145/1553374.1553511.

See Also

Other tree distances: `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`

Examples

```
tree1 <- ape::read.tree(text='((((a, b), c), d), (e, (f, (g, h))));')
tree2 <- ape::read.tree(text='(((a, b), (c, d)), ((e, f), (g, h)));')
tree3 <- ape::read.tree(text='((((h, b), c), d), (e, (f, (g, a))));')

# Best possible score is obtained by matching a tree with itself
DifferentPhylogeneticInfo(tree1, tree1) # 0, by definition
SharedPhylogeneticInfo(tree1, tree1)
SplitwiseInfo(tree1) # Maximum shared phylogenetic information

# Best possible score is a function of tree shape; the splits within
# balanced trees are more independent and thus contain less information
SplitwiseInfo(tree2)

# How similar are two trees?
SharedPhylogeneticInfo(tree1, tree2) # Amount of phylogenetic information in common
VisualizeMatching(SharedPhylogeneticInfo, tree1, tree2) # Which clades are matched?

DifferentPhylogeneticInfo(tree1, tree2) # Distance measure
DifferentPhylogeneticInfo(tree2, tree1) # The metric is symmetric

# Are they more similar than two trees of this shape would be by chance?
ExpectedVariation(tree1, tree2, sample=12)['DifferentPhylogeneticInfo', 'Estimate']

# Every split in tree1 conflicts with every split in tree3
# Pairs of conflicting splits contain clustering, but not phylogenetic,
# information
SharedPhylogeneticInfo(tree1, tree3) # = 0
MutualClusteringInfo(tree1, tree3) # > 0

# Converting trees to Splits objects can speed up multiple comparisons
splits1 <- TreeTools::as.Splits(tree1)
splits2 <- TreeTools::as.Splits(tree2)

SharedPhylogeneticInfoSplits(splits1, splits2)
MatchingSplitInfoSplits(splits1, splits2)
MutualClusteringInfoSplits(splits1, splits2)
```

`VisualizeMatching` *Visualise a matching*

Description

Depict the splits that are matched between two trees using a specified Generalized Robinson-Foulds similarity measure.

Usage

```
VisualizeMatching(
  Func,
  tree1,
  tree2,
  setPar = TRUE,
  precision = 3L,
  Plot = plot.phylo,
  matchZeros = TRUE,
  plainEdges = FALSE,
  edge.width = 1,
  edge.color = "black",
  ...
)
```

Arguments

`Func`	Function used to construct tree similarity.
`tree1, tree2`	Trees of class `phylo`, with identical leaf labels.
`setPar`	Logical specifying whether graphical parameters should be set to display trees side by side.
`precision`	Integer specifying number of significant figures to display when reporting matching scores.
`Plot`	Function to use to plot trees.
`matchZeros`	Logical specifying whether to pair splits with zero similarity (`TRUE`), or leave them unpaired (`FALSE`).
`plainEdges`	Logical specifying whether to plot edges with a uniform width and colour (`TRUE`), or whether to draw edge widths according to the similarity of the associated splits (`FALSE`).
`edge.width, edge.color, ...`	Additional parameters to send to `Plot()`.

Details

Note that when visualizing a Robinson-Foulds distance (using `Func = RobinsonFouldsMatching`), matched splits are assigned a *similarity* score of 1, which is deducted from the total number of splits to calculate the Robinson-Foulds *distance*. Unmatched splits thus contribute one to total tree distance.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

```
tree1 <- TreeTools::BalancedTree(6)
tree2 <- TreeTools::PectinateTree(6)

VisualizeMatching(RobinsonFouldsMatching, tree1, tree2)
VisualizeMatching(SharedPhylogeneticInfo, tree1, tree2, matchZeros = FALSE)
```

Index