Package: TreeDist (via r-universe)

October 11, 2026

Title Calculate and Map Distances Between Phylogenetic Trees

Version 2.15.0

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>; the Hierarchical Mutual Information (Perotti et al. 2015) <doi:10.1103/PhysRevE.92.062825>; 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>. Includes tools for visualizing mappings of tree space (Smith 2022) <doi:10.1093/sysbio/syab100>, for identifying islands of trees (Silva and Wilkinson 2021) <doi:10.1093/sysbio/syab015>, for calculating the median of sets of trees, and for computing the information content of trees and splits.

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

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

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

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

Depends R (>= 4.0), stats,

Imports ape (>= 5.0), cli (>= 3.0), Rdpack (>= 0.7), shiny, shinyjs, TreeTools (>= 2.1.0),

Suggests bookdown, cluster, ggplot2, hypervolume, kdensity, knitr, MASS, phangorn (>= 2.2.1), plotly, PlotTools, protoclust, Quartet, readxl, rmarkdown, Rcpp (>= 1.0.8), rgl, Rogue, spelling, TBRDist, testthat (>= 3.0), Ternary (>= 1.1.2), TreeDistData (> 0.1.0), TreeSearch (>= 1.4.0), Umatrix, vdiffr (>= 1.0.0), withr,

LinkingTo Rcpp, TreeTools (>= 2.1.0),

RdMacros Rdpack

VignetteBuilder knitr

Config/Needs/app/optional uwot

Config/Needs/check rcmdcheck

Config/Needs/coverage covr

Config/Needs/memcheck pkgdown, testthat

Config/Needs/metadata codemetar

Config/Needs/revdeps revdepcheck

Config/Needs/website openssl, pkgdown, remotes, shinylive

Config/roxygen2/version 8.1.0

Config/testthat/parallel false

Config/testthat/edition 3

SystemRequirements C++17, pandoc-citeproc

ByteCompile true

Encoding UTF-8

Language en-GB

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

NeedsCompilation yes

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

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

Repository https://cran.r-universe.dev

Date/Publication 2026-09-11 14:40:02 UTC

RemoteUrl https://github.com/cran/TreeDist

RemoteRef HEAD

RemoteSha 489e6d8ae15f6b93f843e361e2786b70f16be409

Contents

AllSplitPairings . . . 4
cluster-statistics . . . 5
CompareAll . . . 6
Entropy . . . 7
HierarchicalMutualInfo . . . 8
HPart . . . 11
Islands . . . 12
JaccardRobinsonFoulds . . . 13
KendallColijn . . . 16
KMeansPP . . . 18
LAPJV . . . 20
MappingQuality . . . 21
MapTrees . . . 22
MASTSize . . . 24
MatchingSplitDistance . . . 26
MCITree . . . 28
median.multiPhylo . . . 29
MeilaVariationOfInformation . . . 30
MSTSegments . . . 32
NNIDist . . . 33
NNIMaxStep . . . 36
NyeSimilarity . . . 37
PathDist . . . 39
Plot3 . . . 41
ReduceTrees . . . 42
Robinson-Foulds . . . 43
SpectralEigens . . . 46
SplitEntropy . . . 47
SplitsCompatible . . . 48
SplitSharedInformation . . . 49
SPRDist . . . 51
StartParallel . . . 52
TransferConsensus . . . 54
TransferDist . . . 56
TreeDistance . . . 58
TreeInfo . . . 63
VisualizeMatching . . . 67

Index **70**

`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 (Smith 2020), 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

Meila 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*, **36**(20), 5007–5013. doi:10.1093/bioinformatics/btaa614.

Examples

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

cluster-statistics *Cluster size statistics*

Description

Cluster size statistics

Usage

```
SumOfRanges(x, cluster = 1)

SumOfVariances(x, cluster = 1)

SumOfVars(x, cluster = 1)

MeanCentroidDistance(x, cluster = 1, Average = mean)

MeanCentDist(x, cluster = 1, Average = mean)

MeanCentroidDist(x, cluster = 1, Average = mean)

DistanceFromMedian(x, cluster = 1, Average = mean)

DistFromMed(x, cluster = 1, Average = mean)

MeanNN(x, cluster = 1, Average = mean)

MeanMSTEdge(x, cluster = 1)
```

Arguments

x	Matrix in which each row lists the coordinates of a point in a Euclidian space; or, where supported, `dist` object specifying distances between each pair of points.
cluster	Optional integer vector specifying the cluster or group to which each row in `x` belongs.
Average	Function to use to summarize distances. Defaults to `mean`; specifying `median` returns a value akin to the median absolute divergence (see `mad`).

Value

`SumOfRanges()` returns a numeric specifying the sum of ranges within each cluster across all dimensions.

`SumOfVariances()` returns a numeric specifying the sum of variances within each cluster across all dimensions.

`MeanCentroidDistance()` returns a numeric specifying the mean distance from the centroid to points in each cluster.

`DistanceFromMedian()` returns a numeric specifying the mean distance of each point (except the median) from the median point of its cluster.

`MeanNN()` returns a numeric specifying the mean distance from each point within a cluster to its nearest neighbour.

`MeanMSTEdge()` returns a numeric specifying the mean length of an edge in the minimum spanning tree of points within each cluster.

Author(s)

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

See Also

Other tree space functions: `Islands()`, `MSTSegments()`, `MapTrees()`, `MappingQuality()`, `SpectralEigens()`, `median.multiPhylo()`

Other cluster functions: `KMeansPP()`

Examples

```
points <- rbind(matrix(1:16, 4), rep(1, 4), matrix(1:32, 8, 4) / 10)
cluster <- rep(1:3, c(4, 1, 8))

plot(
  points[, 1:2], # Plot first two dimensions of four-dimensional space
  col = cluster, pch = cluster, # Style by cluster membership
  asp = 1, # Fix aspect ratio to avoid distortion
  ann = FALSE, frame = FALSE # Simple axes
)

SumOfRanges(points, cluster)
SumOfVariances(points, cluster)
MeanCentroidDistance(points, cluster)
DistanceFromMedian(points, cluster)
MeanNN(points, cluster)
MeanMSTEdge(points, cluster)
```

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)
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(...)

Ntropy(...)
```

Arguments

...	Series of numerics, or single numeric vector, specifying probabilities of outcomes (for `Entropy()`) or counts (for `Ntropy()`).

Value

`Entropy()` and `Ntropy()` return the entropy of the specified probabilities or counts, 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
Ntropy(c(2, 2, 0, 2, 2)) # = 2
```

`HierarchicalMutualInfo`

Hierarchical Mutual Information

Description

Calculate the Hierarchical Mutual Information (HMI) between two trees, following the recursive algorithm of Perotti et al. (2020).

This function was written during a code sprint: its documentation and test cases have not yet been carefully scrutinized, and its implementation may change without notice. Please alert the maintainer to any issues you encounter.

Usage

```
HierarchicalMutualInfo(tree1, tree2 = NULL, normalize = FALSE)

HMI(tree1, tree2 = NULL, normalize = FALSE)

SelfHMI(tree)

EHMI(tree1, tree2, precision = 0.01, minResample = 36)

AHMI(tree1, tree2, Mean = max, precision = 0.01, minResample = 36)
```

Arguments

`normalize`	If `FALSE`, return the raw HMI, in bits. If `TRUE`, normalize to range [0,1] by dividing by `max(SelfHMI(tree1), SelfHMI(tree2))`. If a function, divide by `normalize(SelfHMI(tree1), SelfHMI(tree2))`.
`tree, tree1, tree2`	An object that can be coerced to an `HPart` object.
`precision`	Numeric; Monte Carlo sampling will terminate once the relative standard error falls below this value.
`minResample`	Integer specifying minimum number of Monte Carlo samples to conduct. Avoids early termination when sample size is too small to reliably estimate the standard error of the mean.
`Mean`	Function by which to combine the self-information of the two input hierarchies, in order to normalize the HMI.

Details

`HierarchicalMutualInfo()` computes the hierarchical mutual content of trees (Perotti et al. 2015; Perotti et al. 2020), which accounts for the non-independence of information represented by nested splits.

`tree` is converted to a set of hierarchical partitions, and the mutual information (in bits) is computed recursively; the contribution of a node is given by:

$$I(t,s) = \log_2(n_{ts}) - \frac{H_{us} + H_{tv} - H_{uv}}{n_{ts}} + \text{mean}(I_{uv})$$

Where:

- n_{ts} is the number of common elements between partitions
- H_{us}, H_{tv}, H_{uv} are entropy terms from child comparisons
- I_{uv} is the recursive HMI for child pairs

`AHMI()` calculates the adjusted hierarchical mutual information:

$$\text{AHMI}(t,s) = \frac{I(t,s) - \hat{I}(t,s)}{\text{mean}(H(t), H(s)) - \hat{I}(t,s)}$$

Where:

- $I(t,s)$ is the hierarchical mutual information between `tree1` and `tree2`
- $\hat{I}(t,s)$ is the expected HMI between `tree1` and `tree2`, estimated by Monte Carlo sampling
- $H(t), H(s)$ is the entropy (self-mutual information) of each tree

Value

`HierarchicalMutualInfo()` returns a numeric value representing the hierarchical mutual information between the input trees, in bits, normalized as specified. Higher values indicate more shared hierarchical structure.

`SelfHMI()` returns the hierarchical mutual information of a tree compared with itself, i.e. its hierarchical entropy (HH).

`EHMI()` returns the expected HMI against a uniform shuffling of element labels, estimated by performing Monte Carlo resampling on the same hierarchical structure until the relative standard error of the estimate falls below `precision`. The attributes of the returned object list the variance (`var`), standard deviation (`sd`), standard error of the mean (`sem`) and relative error (`relativeError`) of the estimate, and the number of Monte Carlo samples used to obtain it (`samples`).

`AHMI()` returns the adjusted HMI, normalized such that zero corresponds to the expected HMI given a random shuffling of elements on the same hierarchical structure. The attribute `sem` gives the standard error of the estimate.

References

Perotti JI, Almeira N, Saracco F (2020). "Towards a Generalization of Information Theory for Hierarchical Partitions." *Physical Review E*, **101**(6), 062148. doi:10.1103/PhysRevE.101.062148.

Perotti JI, Tessone CJ, Caldarelli G (2015). "Hierarchical Mutual Information for the Comparison of Hierarchical Community Structures in Complex Networks." *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, **92**(6), 062825-1–062825-13. doi:10.1103/PhysRevE.92.062825.

See Also

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

Examples

```
library("TreeTools", quietly = TRUE)

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

# Calculate HMI between two trees
HierarchicalMutualInfo(tree1, tree2)

# HMI normalized against the mean information content of tree1 and tree2
HierarchicalMutualInfo(tree1, tree2, normalize = mean)

# Normalized HMI above is equivalent to:
HMI(tree1, tree2) / mean(SelfHMI(tree1), SelfHMI(tree2))
# Expected mutual info for this pair of hierarchies
EHMI(tree1, tree2, precision = 0.1)
# The adjusted HMI normalizes against this expectation
AHMI(tree1, tree2, precision = 0.1)
```

HPart *Hierarchical partition structure*

Description

A structure of class HPart comprises a pointer to a C++ representation of hierarchical partitions, with the attribute tip.label recording the character labels of its leaves. HPart objects with identical tip labels can be compared using HierarchicalMutualInfo().

Usage

```
as.HPart(tree, tipLabels)

## S3 method for class 'HPart'
as.HPart(tree, tipLabels = NULL)

## Default S3 method:
as.HPart(tree, tipLabels = NULL)

## S3 method for class 'list'
as.HPart(tree, tipLabels = NULL)

## S3 method for class 'phylo'
as.HPart(tree, tipLabels = TipLabels(tree))

is.HPart(x)

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

## S3 method for class 'HPart'
as.phylo(x, ...)

## S3 method for class 'HPart'
plot(x, ...)
```

Arguments

tree	An object to convert to an HPart structure, in a supported format (see details).
tipLabels	Character vector specifying sequence in which to order tip labels.
x	HPart object to plot.
...	Additional arguments to plot.phylo.

Details

An HPart object may be created from various representations of hierarchical structures:

- a tree (possibly phylogenetic) of class `phylo`
- A hierarchical list of lists, in which elements are represented by integers 1 to n
- A vector, which will be interpreted as a flat structure in which all elements bearing the same label are assigned to the same cluster

Value

`HPart()` returns a structure containing a pointer to a C++ representation of a hierarchical partition structure.

`Islands` *Find islands from distance matrix*

Description

`Islands()` assigns a set of objects to islands, such that all elements within an island can form a connected graph in which each edge is no longer than `threshold` distance units (Silva and Wilkinson 2021).

Usage

```
Islands(D, threshold, dense = TRUE, smallest = 0)
```

Arguments

`D`	Square matrix or `dist` object containing Euclidean distances between data points.
`threshold`	Elements greater than `threshold` distance units will not be assigned to the same island.
`dense`	Logical; if `FALSE`, each island will be named according to the index of its lowest-indexed member; if `TRUE`, each island will be numbered sequentially from `1`, ordered by the index of the lowest-indexed member.
`smallest`	Integer; Islands comprising no more than `smallest` elements will be assigned to island `NA`.

Value

`Islands()` returns a vector listing the island to which each element is assigned.

Author(s)

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

References

Silva AS, Wilkinson M (2021). "On Defining and Finding Islands of Trees and Mitigating Large Island Bias." *Systematic Biology*, **70**(6), 1282–1294. doi:10.1093/sysbio/syab015.

See Also

Other tree space functions: MSTSegments(), MapTrees(), MappingQuality(), SpectralEigens(), cluster-statistics, median.multiPhylo()

Examples

```
library("TreeTools", quietly = TRUE)
# Generate a set of trees
trees <- as.phylo(as.TreeNumber(BalancedTree(16)) + c(-(40:20), 70:105), 16)

# Calculate distances between trees
distances <- ClusteringInfoDist(trees)
summary(distances)

# Assign trees to islands
isle <- Islands(distances, quantile(distances, 0.1))
table(isle)

# Indicate island membership on 2D mapping of tree distances
mapping <- cmdscale(distances, 2)
plot(mapping, col = isle + 1,
     asp = 1, # Preserve aspect ratio - do not distort distances
     ann = FALSE, axes = FALSE, # Don't label axes: dimensions are meaningless)
     pch = 16 # Plotting character: Filled circle
)

# Compare strict consensus with island consensus trees
oPar <- par(mfrow = c(2, 2), mai = rep(0.1, 4))
plot(Consensus(trees), main = "Strict")
plot(Consensus(trees[isle == 1]), edge.col = 2, main = "Island 1")
plot(Consensus(trees[isle == 2]), edge.col = 3, main = "Island 2")
plot(Consensus(trees[isle == 3]), edge.col = 4, main = "Island 3")

# Restore graphical parameters
par(oPar)
```

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 = NULL,
```

```
  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, tree2	Trees of class phylo, with leaves labelled identically, or lists of such trees to undergo pairwise comparison. Where implemented, tree2 = NULL will compute distances between each pair of trees in the list tree1 using a fast algorithm based on Day (1985).
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, 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.

Trees need not contain identical leaves; scores are based on the leaves that trees hold in common. Check for unexpected differences in tip labelling with `setdiff(TipLabels(tree1), TipLabels(tree2))`.

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

Böcker 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.

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

Nye TMW, Liò 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: `HierarchicalMutualInfo()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `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 = NULL, Vector = KCVector)

KCVector(tree)

PathVector(tree)

SplitVector(tree)

KCDiameter(tree)
```

Arguments

`tree1`, `tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).
`Vector`	Function converting a tree to a numeric vector. `KCVector`, the default, returns the number of edges between the common ancestor of each pair of leaves and the root of the tree (per Kendall and Colijn 2016). `PathVector` returns the number of edges between each pair of leaves (per Steel and Penny 1993). `SplitVector` returns the size of the smallest split that contains each pair of leaves (per Smith 2022).
`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 Euclidean 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.

An analogous distance can be created from any vector representation of a tree. The split size vector metric (Smith 2022) is an attempt to mimic the Kendall Colijn metric in situations where the position of the root should not be afforded special significance; and the path distance (Steel and Penny 1993) is a familiar alternative whose underlying vector 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.

None of these vector-based methods performs as well as other tree distances in measuring similarities in the relationships implied by a pair of trees (Smith 2020); in particular, the Kendall Colijn metric is strongly influenced by tree balance, and may not be appropriate for a suite of common applications (Smith 2022).

Value

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

`KCDiameter()` returns the value of the Kendall & Colijn's (2016) metric distance between two pectinate trees with n leaves ordered in the opposite direction, which I suggest (without any attempt at a proof) may be a useful proxy for the diameter (i.e. maximum value) of the K–C metric.

Functions

- `KCVector()`: Creates a vector that characterises a rooted tree, as described in Kendall and Colijn (2016).
- `PathVector()`: Creates a vector reporting the number of edges between each pair of leaves, per the path metric of Steel and Penny (1993).
- `SplitVector()`: Creates a vector reporting the smallest split containing each pair of leaves, per the metric proposed in Smith (2022).

Author(s)

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

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

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.

Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing

phylogenetic trees." *Bioinformatics*, **36**(20), 5007–5013. doi:10.1093/bioinformatics/btaa614.

Smith MR (2022). "Robust analysis of phylogenetic tree space." *Systematic Biology*, **71**(5), 1255–1270. doi:10.1093/sysbio/syab100.

Steel MA, Penny D (1993). "Distributions of tree comparison metrics—some new results." *Systematic Biology*, **42**(2), 126–141. doi:10.1093/sysbio/42.2.126.

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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `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))

KendallColijn(lapply(rep(8, 4), ape::rtree), Vector = SplitVector)

# Notice that changing tree shape close to the root results in much
# larger differences
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 = "(a, (b, (c, (d, (e, ((f, g), h))))));")
trees <- c(tree1, tree2, tree3)
KendallColijn(trees)
KendallColijn(trees, Vector = SplitVector)
KCDiameter(4)
KCDiameter(trees)
```

`KMeansPP` *k-means++ clustering*

Description

k-means++ clustering (Arthur and Vassilvitskii 2007) improves the speed and accuracy of standard `kmeans` clustering (Hartigan and Wong 1979) by preferring initial cluster centres that are far from others. A scalable version of the algorithm has been proposed for larger data sets (Bahmani et al. 2012), but is not implemented here.

Usage

```
KMeansPP(x, k = 2, nstart = 10, ...)
```

Arguments

x	Numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
k	Integer specifying the number of clusters, *k*.
nstart	Positive integer specifying how many random sets should be chosen
...	additional arguments passed to kmeans

Author(s)

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

References

Arthur D, Vassilvitskii S (2007). "K-Means++: The Advantages of Careful Seeding." In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, 1027–1035.

Bahmani B, Moseley B, Vattani A, Kumar R, Vassilvitskii S (2012). "Scalable K-Means++." *arXiv*. doi:10.48550/arXiv.1203.6402. 1203.6402.

Hartigan JA, Wong MA (1979). "Algorithm AS 136: a *K*-means clustering algorithm." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **28**(1), 100–108. doi:10.2307/2346830.

See Also

kmeans

Other cluster functions: cluster-statistics

Examples

```
# Generate random points
set.seed(1)
x <- cbind(c(rnorm(10, -5), rnorm(5, 1), rnorm(10, 6)),
           c(rnorm(5, 0), rnorm(15, 4), rnorm(5, 0)))

# Conventional k-means may perform poorly
klusters <- kmeans(x, cent = 5)
plot(x, col = klusters$cluster, pch = rep(15:19, each = 5))

# Here, k-means++ recovers a better clustering
plusters <- KMeansPP(x, k = 5)
plot(x, col = plusters$cluster, pch = rep(15:19, each = 5))
```

LAPJV *Solve linear assignment problem using LAPJV*

Description

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

Usage

```
LAPJV(x)
```

Arguments

x	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.

Value

`LAPJV()` returns 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.

See Also

Implementations of the Hungarian algorithm exist in **adagio**, **RcppHungarian**, and **clue** and **lpSolve**; for larger matrices, these are substantially slower. (See discussion at Stack Overflow.)

The JV algorithm is implemented for square matrices in the Bioconductor package `GraphAlignment::LinearAssignment()`

Examples

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

LAPJV(problem)
```

`MappingQuality` *Faithfulness of mapped distances*

Description

`MappingQuality()` calculates the trustworthiness and continuity of mapped distances (Venna and Kaski 2001; Kaski et al. 2003). Trustworthiness measures, on a scale from 0–1, the degree to which points that are nearby in a mapping are truly close neighbours; continuity, the extent to which points that are truly nearby retain their close spatial proximity in a mapping.

Usage

```
MappingQuality(original, mapped, neighbours = 10L)

ProjectionQuality(original, mapped, neighbours = 10L)
```

Arguments

`original`, `mapped`	Square matrix or `dist` object containing original / mapped pairwise distances.
`neighbours`	Integer specifying number of nearest neighbours to use in calculation. This should typically be small relative to the number of points.

Value

`MappingQuality()` returns a named vector of length four, containing the entries: `Trustworthiness`, `Continuity`, `TxC` (the product of these values), and `sqrtTxC` (its square root).

Author(s)

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

References

Kaski S, Nikkila J, Oja M, Venna J, Toronen P, Castren E (2003). "Trustworthiness and metrics in visualizing similarity of gene expression." *BMC Bioinformatics*, **4**, 48. doi:10.1186/14712105448.

Venna J, Kaski S (2001). "Neighborhood preservation in nonlinear projection methods: an experimental study." In Dorffner G, Bischof H, Hornik K (eds.), *Artificial Neural Networks — ICANN 2001*, Lecture Notes in Computer Science, 485–491. doi:10.1007/3540446680_68.

See Also

Other tree space functions: `Islands()`, `MSTSegments()`, `MapTrees()`, `SpectralEigens()`, `cluster-statistics`, `median.multiPhylo()`

Examples

```
library("TreeTools", quietly = TRUE)
trees <- as.phylo(0:10, nTip = 8)
distances <- ClusteringInfoDistance(trees)
mapping <- cmdscale(distances)
MappingQuality(distances, dist(mapping), 4)
```

MapTrees *Graphical user interface for mapping distances and analysing tree space*

Description

`MapTrees()` launches a "Shiny" application for the visualization and evaluation of tree spaces.

Usage

```
MapTrees()

Project()
```

Input tab

The input tab allows for the upload of sets of phylogenetic trees from file. Trees at the start or end of a file can be excluded, and the number of trees can be brought down to a manageable number by uniformly subsampling every _n_th tree. Samples of c. 100 trees can be analysed in seconds; analysis of larger samples will take longer, particularly with slower methods (e.g. quartet distances; Kruskal-1 MDS; large minimum spanning trees).

Different batches can be plotted with different colours / symbols.

If each tree is associated with a property – for example, the data or method used to generate it, or its stratigraphic congruence – a list of properties for each tree, with one entry per line/row, can be uploaded along with the trees. Points in tree space can then be styled according to the corresponding property.

If trees are subsampled (using the "Sample every" slider), then the values in the tree properties file can also be subsampled accordingly. Unfortunately there is not yet support for multiple point property files; one file will be applied to all trees, in the sequence that they were added to memory.

Analysis tab

Select from a suite of distance methods: clustering information and phylogenetic information are quick and satisfactory; quartet is slow but gives slightly better mappings; path is very fast but may not reflect evolutionary signal very well; and Robinson–Foulds should probably never be used for analysis; it is included for comparison purposes.

Principle components mappings should suffice for most purposes; Sammon and Kruskal mappings are slower and seldom differ by much, in character or quality, but may emphasize outliers more.

Partitioning around medoids or minimax-linkage hierarchical clustering will typically find a close-to-optimal clustering where one exists; select additional methods for a more exhaustive search. To avoid redundant calculation, clusterings are only updated when "recalculate clustering" is clicked, or the "maximum cluster number" slider is modified; clustering solutions using more than this many clusters are not considered Clusterings with silhouette coefficients < 0.25 are unlikely to represent genuine structure and are not reported or depicted.

Display tab

Up to 15 dimensions can be depicted; the quality of a mapping – that is, the faithfulness of mapped distances to true tree-to-tree distances – is quantified by the product of the Trustworthiness and Continuity metrics, which should exceed 0.9 (at least).

An interactive 3D plot can be explored by dragging the mouse and scrolling, but do be careful to check that three dimensions are enough to depict your data accurately.

The minimum spanning tree is the shortest possible line selecting the chosen subsample of trees; if it takes a convoluted zig-zagging route, then the mapping is doing a poor job of reflecting true tree to tree distances.

Convex hulls are the smallest polygons enclosing all points in each cluster; they are handy for spotting clusters, but their area does not correspond to a genuine quantity, so should not be interpreted.

Tree numbers correspond to the sequence of trees in their original input file, before subsampling.

Each tree is denoted by a point, whose symbol can be styled according to cluster membership or according to the file that contains the tree, with each click of "Add to existing" on the input tab constituting a new batch with a new symbol.

Points can be coloured according to a category – the cluster or batch to which they belong, or custom data provided in the Point Property File on the input tab – or continuously, either by the sequence in which they were added to memory, or according to custom data.

Exporting tree spaces

A mapping can be saved to PDF or as a PNG bitmap at the size selected.

References

A list of references employed when constructing the tree space is populated according to the methods used; it would be appropriate to cite and briefly discuss these studies in any publication using

figures generated using this application. The application itself can be cited using Smith (2020, 2022)

Author(s)

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

References

Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing phylogenetic trees." *Bioinformatics*, **36**(20), 5007–5013. doi:10.1093/bioinformatics/btaa614.

Smith MR (2022). "Robust analysis of phylogenetic tree space." *Systematic Biology*, **71**(5), 1255–1270. doi:10.1093/sysbio/syab100.

See Also

Full detail of tree space analysis in R is provided in the accompanying vignette.

Other tree space functions: `Islands()`, `MSTSegments()`, `MappingQuality()`, `SpectralEigens()`, `cluster-statistics`, `median.multiPhylo()`

`MASTSize` *Maximum Agreement Subtree size*

Description

Calculate the size or phylogenetic information content (Steel and 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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `KendallColijn()`, `MatchingSplitDistance()`, `NNIDist()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `TreeDistance()`

Examples

```
 # for as.phylo, BalancedTree, PectinateTree:
library("TreeTools", quietly = TRUE)

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 = NULL,
  normalize = FALSE,
  reportMatching = FALSE
)

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

Arguments

`tree1`, `tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).
`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`, `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

Trees need not contain identical leaves; scores are based on the leaves that trees hold in common. Check for unexpected differences in tip labelling with `setdiff(TipLabels(tree1), TipLabels(tree2))`.

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.

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `NNIDist()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `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))
```

MCITree *Maximum Clade Information Tree*

Description

Analogous to the Maximum Clade Credibility tree: select the tree from a posterior distribution whose clades have the highest information content. Generate the MCC tree by specifying `info = "credibility"`.

Usage

```
MCITree(trees, info = "phylogenetic", check.tips = TRUE)
```

Arguments

`trees`	List of `phylo` objects, optionally with class `multiPhylo`.
`info`	Abbreviation of "phylogenetic" or "clustering", specifying the concept of information to employ.
`check.tips`	Logical specifying whether to renumber leaves such that leaf numbering is consistent in all trees.

Value

`MCITree()` returns the tree with the highest information content, selected from `trees`.

Author(s)

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

See Also

Other summary trees: `TransferConsensus()`

Examples

```
library("TreeTools", quietly = TRUE)
trees <- as.phylo(24:40, 16)

# Maximum Clade Information tree
mci <- MCITree(trees)
SplitwiseInfo(mci)
plot(mci)
p <- SplitFrequency(mci, trees) / length(trees)
LabelSplits(mci, round(p * 100), "%", bg = SupportColor(p))


# Compare with Maximum Clade Credibility tree
mcc <- MCITree(trees, "credibility")
```

```
plot(mcc)
p <- SplitFrequency(mcc, trees) / length(trees)
LabelSplits(mcc, round(p * 100), "%", bg = SupportColor(p))
SplitwiseInfo(mcc)
```

`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()`

Other tree space functions: `Islands()`, `MSTSegments()`, `MapTrees()`, `MappingQuality()`, `SpectralEigens()`, `cluster-statistics`

Examples

```
library("TreeTools", quietly = TRUE)
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

Meila 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))
```

`MSTSegments` *Add minimum spanning tree to plot, colouring by stress*

Description

To identify strain in a multidimensional scaling of distances, it can be useful to plot a minimum spanning tree (Gower 1966; Smith 2022). Colouring each edge of the tree according to its strain can identify areas where the mapping is stretched or compressed.

Usage

```
MSTSegments(mapping, mstEnds, ...)

StrainCol(
  distances,
  mapping,
  mstEnds = MSTEdges(distances),
  palette = rev(hcl.colors(256L, "RdYlBu"))
)
```

Arguments

`mapping`	Two-column matrix giving *x* and *y* coordinates of plotted points.
`mstEnds`	Two-column matrix identifying rows of `mapping` at end of each edge of the MST, as output by `TreeTools::MSTEdges()`.
`...`	Additional arguments to `segments()`.
`distances`	Matrix or `dist` object giving original distances between each pair of points.
`palette`	Vector of colours with which to colour edges.

Value

`StrainCol()` returns a vector in which each entry is selected from `palette`, with an attribute `logStrain` denoting the logarithm of the mapped over original distance, shifted such that the median value is zero. Palette colours are assigned centred on the median value, with entries early in `palette` assigned to edges in which the ratio of mapped distance to original distance is small.

Author(s)

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

References

Gower JC (1966). "Some distance properties of latent root and vector methods used in multivariate analysis." *Biometrika*, **53**(3/4), 325–338. doi:10.2307/2333639.

Smith MR (2022). "Robust analysis of phylogenetic tree space." *Systematic Biology*, **71**(5), 1255–1270. doi:10.1093/sysbio/syab100.

See Also

Other tree space functions: `Islands()`, `MapTrees()`, `MappingQuality()`, `SpectralEigens()`, `cluster-statistics`, `median.multiPhylo()`

Examples

```
set.seed(0)
library("TreeTools", quietly = TRUE)
distances <- ClusteringInfoDist(as.phylo(5:16, 8))
mapping <- cmdscale(distances, k = 2)
mstEnds <- MSTEdges(distances)

# Set up blank plot
plot(mapping, asp = 1, frame.plot = FALSE, ann = FALSE, axes = FALSE,
     type = "n")
# Add MST
MSTSegments(mapping, mstEnds,
            col = StrainCol(distances, mapping, mstEnds))
# Add points at end so they overprint the MST
points(mapping)
PlotTools::SpectrumLegend(
 "bottomleft",
 legend = c("Extended", "Median", "Contracted"),
 bty = "n",     # No box
 y.intersp = 2, # Expand in Y direction
 palette = hcl.colors(256L, "RdYlBu", rev = TRUE)
)
```

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.

$$n \log 2n + \mathrm{O}(n)$$

;
- `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 DF (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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `TreeDistance()`

Examples

```
library("TreeTools", quietly = TRUE)

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)
```

`NNIMaxStep` *Largest clustering information distance reachable by one nearest neighbour interchange*

Description

`NNIMaxStep()` returns the largest Clustering Information Distance that can separate an n-leaf tree from any tree that differs from it by a single nearest neighbour interchange (NNI) move.

Usage

```
NNIMaxStep(tree, normalize = FALSE)
```

Arguments

`tree`	Tree of class `phylo`, or list of trees of class `list` or `multiPhylo`, or an integer specifying the number of leaves in a tree.
`normalize`	Logical specifying whether to normalize the distance against the summed clustering information of the two trees.

Value

`NNIMaxStep()` returns a numeric vector, one entry per tree (or leaf count), giving the largest attainable distance, in bits when `normalize = FALSE`, or as a fraction in the range [0, 1] when `normalize = TRUE`. `NA` is returned where $n < 4$, as no NNI move exists.

The vector bears attributes `"subtrees"`, giving the sizes of the four subtrees around the moved edge, and `"splits"`, the sizes of the two splits that the move exchanges. Where more than one tree is supplied, each attribute is a list with one entry per tree.

Author(s)

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

See Also

The distance itself: `ClusteringInfoDistance()`

Diameter of the NNI metric: `NNIDiameter()`

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

Examples

```
# Largest clustering information distance from a single NNI move
NNIMaxStep(8)  # exactly two bits for any multiple of four
NNIMaxStep(6)  # a little less otherwise

# Read off the maximizing local topology
m6 <- NNIMaxStep(6)
attr(m6, "subtrees")
attr(m6, "splits")

# Vectorized over leaf counts
NNIMaxStep(4:8)

# Computed for a given tree
library("TreeTools", quietly = TRUE)
NNIMaxStep(BalancedTree(19))

# Normalized
NNIMaxStep(12, normalize = TRUE)
```

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

NyeSplitSimilarity(
  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. Where implemented, tree2 = NULL will compute distances between each pair of trees in the list tree1 using a fast algorithm based on Day (1985).
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, 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.

Trees need not contain identical leaves; scores are based on the leaves that trees hold in common. Check for unexpected differences in tip labelling with `setdiff(TipLabels(tree1), TipLabels(tree2))`.

Author(s)

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

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

Nye TMW, Liò 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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NNIMaxStep()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`, `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 rooted or unrooted 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. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).

Details

This function is a faster alternative to the function `path.dist()` in the phangorn package, which can crash if the internal representation of trees does not conform to certain (unspecified) expectations, and which treats all trees as unrooted.

The path distance is calculated by tabulating the cladistic difference (= topological distance) between each pair of tips in each tree. A precursor to the path distance (Farris 1969) took the mean squared difference between the elements of each tree's tabulation (Farris, 1973); the method used here is that proposed by Steel and Penny (1993), which takes the square root of this sum. Other precursor measures are described in Williams and Clifford (1971) and Phipps (1971).

If a root node is present, trees are treated as rooted. To avoid counting the root edge twice, use `UnrootTree(tree)` before calculating the path 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

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

Farris JS (1969). "A successive approximations approach to character weighting." *Systematic Biology*, **18**(4), 374–385. doi:10.2307/2412182.

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

Phipps JB (1971). "Dendrogram topology." *Systematic Zoology*, **20**(3), 306. doi:10.2307/2412343.

Steel MA, Penny D (1993). "Distributions of tree comparison metrics—some new results." *Systematic Biology*, **42**(2), 126–141. doi:10.1093/sysbio/42.2.126.

Williams WT, Clifford HT (1971). "On the comparison of two classifications of the same set of elements." *Taxon*, **20**(4), 519–522. doi:10.2307/1218253.

See Also

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

Examples

```
library("TreeTools")

# Treating the two edges to the root node as distinct
PathDist(BalancedTree(7), PectinateTree(7))

# Counting those two edges once
PathDist(UnrootTree(BalancedTree(7)), UnrootTree(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))

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

Plot3 *Pseudo-3D plotting*

Description

Plot3() displays three-dimensional data in two dimensions, reflecting the third dimension with point scaling, overlap and fogging. Points with a lower z value are smaller than, fainter than, and overlapped by points with a higher value.

Usage

```
Plot3(
  x,
  y = NULL,
  z = NULL,
  pch = par("pch"),
  col = par("col"),
  bg = NA,
  cex = 1,
  axes = TRUE,
  frame.plot = axes,
  plot.bg = NA,
  fog = 1/2,
  shrink = 1/2,
```

```
  add = FALSE,
  ...
)
```

Arguments

x, y, z	Coordinates of points to plot.
bg, cex, col, pch, add, axes, frame.plot, ...	Parameters passed to plot.default().
plot.bg	Colour with which to fill plot area, used as fog colour.
fog	Numeric from zero (no fading) to one (furthest points are invisible) specifying amount to fade distant points.
shrink	Numeric specifying degree to which size of plotted point should reflect z position. 0 denotes no scaling; if 1, furthest point will have zero size.

Author(s)

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

Examples

```
Plot3(1:10, 1:10, 1:10, cex = 7, pch = 16, axes = FALSE, asp = 1)
# Extreme values of fog and shrink will cause smallest z values to
# become invisible.
Plot3(1:10, 1:10, 1:10, cex = 7, pch = 16, axes = FALSE, asp = 1,
      fog = 1, shrink = 1)
```

ReduceTrees *Collapse areas of agreement between two trees*

Description

ReduceTrees() reduces trees according to the tree reduction rules of Allen and Steel (2001):

- Collapse identical pendant subtrees;
- Compress equivalent internal chains.

Usage

```
ReduceTrees(tree1, tree2, check = TRUE)
```

Arguments

tree1, tree2	Single trees of class phylo to undergo comparison.
check	Logical specifying whether to validate input. Specify FALSE and you will encounter undefined behaviour if trees are not binary phylo objects with identical leaf labels, rooted on leaf 1.

Value

`ReduceTrees()` returns a list of two trees, corresponding to `tree1` and `tree2` after any identical groupings have been collapsed, with tree edges listed in postorder; or `NULL` if the trees are equivalent.

Author(s)

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

Examples

```
tree1 <- TreeTools::BalancedTree(9)
tree2 <- TreeTools::PectinateTree(9)

# Set graphical parameters
oPar <- par(mai = rep(0.1, 4), mfrow = c(2, 2))

plot(tree1)
plot(tree2)

# Reduce trees by collapsing identical clades
confl <- ReduceTrees(tree1, tree2)

plot(confl[[1]])
plot(confl[[2]])

# Restore graphical parameters
par(oPar)
```

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

Description

`RobinsonFoulds()` calculates the Robinson–Foulds distance (Robinson and Foulds 1981), or the corresponding similarity measure. `InfoRobinsonFoulds()` weights splits according to their phylogenetic information content (§2.1 in Smith 2020). Optionally, the matching between identical splits may reported. Generalized Robinson–Foulds distances (see `TreeDistance()`) are better suited to most use cases (Smith 2020, 2022).

Usage

```
InfoRobinsonFoulds(
  tree1,
  tree2 = NULL,
  similarity = FALSE,
  normalize = FALSE,
```

```
  reportMatching = FALSE
)

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

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

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

RobinsonFouldsSplits(
  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. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).
`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`, `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

`RobinsonFoulds()` calculates the standard Robinson–Foulds distance, i.e. the number of splits that occur in one tree but not the other. `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.

Rapid comparison between multiple pairs of trees employs the Day (1985) linear-time algorithm.

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`.

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

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 each tree, treated independently.

Author(s)

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

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

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

Smith MR (2020). "Information theoretic Generalized Robinson-Foulds metrics for comparing phylogenetic trees." *Bioinformatics*, **36**(20), 5007–5013. doi:10.1093/bioinformatics/btaa614.

Smith MR (2022). "Robust analysis of phylogenetic tree space." *Systematic Biology*, **71**(5), 1255–1270. doi:10.1093/sysbio/syab100.

See Also

Display paired splits: `VisualizeMatching()`

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

Examples

```
 # For BalancedTree, PectinateTree, as.phylo:
library("TreeTools", quietly = TRUE)
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)
```

SpectralEigens *Eigenvalues for spectral clustering*

Description

Spectral clustering emphasizes nearest neighbours when forming clusters; it avoids some of the issues that arise from clustering around means / medoids.

Usage

```
SpectralEigens(D, nn = 10L, nEig = 2L)

SpectralClustering(D, nn = 10L, nEig = 2L)
```

Arguments

`D`	Square matrix or `dist` object containing Euclidean distances between data points.
`nn`	Integer specifying number of nearest neighbours to consider
`nEig`	Integer specifying number of eigenvectors to retain.

Value

`SpectralEigens()` returns spectral eigenvalues that can then be clustered using a method of choice.

Author(s)

Adapted by MRS from script by Nura Kawa

See Also

Other tree space functions: `Islands()`, `MSTSegments()`, `MapTrees()`, `MappingQuality()`, `cluster-statistics`, `median.multiPhylo()`

Examples

```
library("TreeTools", quietly = TRUE)
trees <- as.phylo(0:18, nTip = 8)
distances <- ClusteringInfoDistance(trees)
eigens <- SpectralEigens(distances)
# Perform clustering:
clusts <- KMeansPP(dist(eigens), k = 3)
plot(eigens, pch = 15, col = clusts$cluster)
plot(cmdscale(distances), pch = 15, col = clusts$cluster)
```

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.

Meila 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: `SplitSharedInformation()`, `TreeInfo`

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)

Log2TreesConsistentWithTwoSplits(n, A1, A2 = A1)

Log2TreesConsistentWithTwoSplits(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 (Meila 2007), 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()`.
- `Log2TreesConsistentWithTwoSplits()`: Base two logarithm of `TreesConsistentWithTwoSplits()`.
- `Log2TreesConsistentWithTwoSplits()`: Base 2 logarithm of `TreesConsistentWithTwoSplits()`.

Author(s)

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

References

Meila 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: `SplitEntropy()`, `TreeInfo`

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)
```

`SPRDist` *Approximate the Subtree Prune and Regraft distance*

Description

`SPRDist()` approximates the SPR distance between trees.

Usage

```
SPRDist(tree1, tree2 = NULL, method = "deOliveira", symmetric)

## S3 method for class 'phylo'
SPRDist(tree1, tree2 = NULL, method = "deOliveira", symmetric)

## S3 method for class 'list'
SPRDist(tree1, tree2 = NULL, method = "deOliveira", symmetric)

## S3 method for class 'multiPhylo'
SPRDist(tree1, tree2 = NULL, method = "deOliveira", symmetric)
```

Arguments

`tree1`, `tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).
`method`	Character specifying which method to use to approximate the SPR distance. Currently defaults to `"deOliveira"`. "Rogue"` implements an experimental method whose details are pending publication; this function is under development, and may be modified or removed without notice. Once formally validated, it is anticipated that this method will become the default.
`symmetric`	Deprecated (redundant after fix of phangorn#97).

Details

The function currently defaults to the heuristic method of de Oliveira Martins et al. (2008), which purports to provide an upper bound on the SPR distance (though exceptions exist). Other approximations (e.g. Hickey et al. 2008, Goloboff 2008, Whidden and Matsen 2018) are not yet implemented.

Value

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

Author(s)

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

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

Goloboff PA (2008). "Calculating SPR distances between trees." *Cladistics*, **24**(4), 591-597. doi:10.1111/j.10960031.2007.00189.x.

Hickey G, Dehne F, Rau-Chaplin A, Blouin C (2008). "SPR distance computation for *unrooted* trees." *Evolutionary Bioinformatics*, **4**, EBO–S419. doi:10.4137/EBO.S419.

Whidden C, Matsen FA (2018). "Efficiently Inferring Pairwise Subtree Prune-and-Regraft Adjacencies between Phylogenetic Trees." *2018 Proceedings of the Meeting on Analytic Algorithmics and Combinatorics (ANALCO)*, 77–91. doi:10.1137/1.9781611975062.8.

de Oliveira Martins L, Leal E, Kishino H (2008). "Phylogenetic detection of recombination with a Bayesian prior on the distance between trees." *PLoS One*, **3**(7), e2651. doi:10.1371/journal.pone.0002651.

See Also

Exact calculation with **TBRDist** functions `USPRDist()` and `ReplugDist()`.

phangorn function `SPR.dist()` employs the de Oliveira Martins et al. (2008) algorithm but can crash when sent trees of certain formats, and tends to have a longer running time.

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

Examples

```
library("TreeTools", quietly = TRUE)

# Compare single pair of trees
SPRDist(BalancedTree(7), PectinateTree(7))

# Compare all pairs of trees
SPRDist(as.phylo(30:33, 8))

# Compare each tree in one list with each tree in another
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))
```

`StartParallel` *Calculate distances in parallel*

Description

Accelerate distance calculation by employing multiple CPU workers.

Usage

```
StartParallel(...)

SetParallel(cl)

GetParallel(cl)

StopParallel(quietly = FALSE)
```

Arguments

`...`	Parameters to pass to `makeCluster()`.
`cl`	An existing cluster.
`quietly`	Logical; if `TRUE`, do not warn when no cluster was running.

Details

OpenMP (recommended for all split-based metrics):

When the package is built with OPENMP support (the default on Linux and Windows; optional on macOS), all pairwise split-based distance calculations use an efficient multi-threaded batch path automatically — no cluster setup is required. The affected functions are:

- `ClusteringInfoDistance()` / `MutualClusteringInfo()`
- `SharedPhylogeneticInfo()` / `DifferentPhylogeneticInfo()`
- `MatchingSplitInfo()` / `MatchingSplitInfoDistance()`
- `MatchingSplitDistance()`
- `InfoRobinsonFoulds()`
- `NyeSimilarity()`
- `JaccardRobinsonFoulds()`

The number of OPENMP threads is controlled by the standard `"mc.cores"` option:

```
options(mc.cores = parallel::detectCores())  # use all available cores
options(mc.cores = 4L)                        # or a fixed number
```

The default is 1 (single-threaded).

R parallel cluster:

`StartParallel()` creates an R socket cluster (via `makeCluster()`) and registers it for use by TreeDist. `SetParallel()` registers a pre-existing cluster. `StopParallel()` stops the cluster and releases resources.

When to use `StartParallel()`: for metrics that do not have an OPENMP batch path, namely tree-object-based distances such as `NNIDist()` and `MASTSize()` / `MASTInfo()`, or any function called via `CompareAll()`. R-cluster parallelism carries a serialisation overhead of ~2–3 s, so it is only beneficial for large problems.

When *not* to use `StartParallel()`: for the split-based metrics listed above. Registering a cluster disables the OPENMP batch path for those functions, replacing a thread-local C++ loop with inter-process communication — which is slower at every problem size measured. Call `StopParallel()` before computing split-based distances if a cluster is active.

Value

`StartParallel()` and `SetParallel()` return the previous value of `options("TreeDist-cluster")`.

`GetParallel()` returns the currently specified cluster.

`StopParallel()` returns `TRUE` if a cluster was destroyed, `FALSE` otherwise.

Author(s)

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

Examples

```
# OpenMP parallelism: set mc.cores before calling any split-based metric.
options(mc.cores = 2L)
# MutualClusteringInfo(trees)  # uses 2 OpenMP threads automatically
options(mc.cores = NULL)  # restore default (single-threaded)

if (interactive()) {
  # R cluster: beneficial for NNIDist, MASTSize/MASTInfo, CompareAll(), etc.
  # Do NOT activate while computing split-based distances (MCI, SPI, MSI, . . . )
  # as it bypasses the faster OpenMP path.
  library("TreeTools", quietly = TRUE)
  nCores <- ceiling(parallel::detectCores() / 2)
  StartParallel(nCores) # Takes a few seconds to set up processes
  GetParallel()
  CompareAll(as.phylo(0:6, 100), NNIDist)
  StopParallel() # Returns system resources
}
```

TransferConsensus *Consensus tree minimizing transfer distance*

Description

Construct a consensus tree that minimizes the sum of transfer distances to a set of input trees, using a greedy add-and-prune heuristic. This function has moved to `ConsTree::Transfer()` and will soon be removed.

Usage

```
TransferConsensus(
  trees,
  scale = TRUE,
  greedy = c("best", "first"),
  init = c("empty", "majority")
)
```

Arguments

trees	An object of class multiPhylo: the input trees. All trees must share the same tip labels.
scale	Logical; if TRUE (default), use the scaled transfer distance (normalized by light-side size minus one). If FALSE, use the unscaled (raw Hamming) transfer distance.
greedy	Character string specifying the greedy strategy: "best" (default) picks the single highest-benefit action at each step; "first" picks the first improving action encountered (faster but potentially lower quality).
init	Character string specifying the initial consensus: "empty" (default) starts with no splits (purely additive); "majority" starts with the majority-rule consensus and refines.

Details

Unlike the majority-rule consensus, which minimizes Robinson-Foulds distance and can be highly unresolved when phylogenetic signal is low, TransferConsensus() uses the finer-grained transfer distance (Lemoine et al. 2018) to construct a more resolved consensus tree.

The algorithm pools all splits observed across input trees, computes pairwise transfer distances between them, and greedily adds or removes splits to minimize total transfer dissimilarity cost. The approach follows Takazawa et al. (2026), reimplemented for 'TreeDist' infrastructure.

Value

A tree of class phylo.

References

Lemoine F, Domelevo Entfellner J, Wilkinson E, Correia D, Dávila Felipe M, De Oliveira T, Gascuel O (2018). "Renewing Felsenstein's phylogenetic bootstrap in the era of big data." *Nature*, **556**(7702), 452–456. doi:10.1038/s41586-018-0043-0.

Takazawa Y, Takeda A, Hayamizu M, Gascuel O (2026). "Outperforming the majority-rule consensus tree using fine-grained dissimilarity measures." *bioRxiv*. doi:10.64898/2026.03.16.712085.

See Also

Other summary trees: MCITree()

`TransferDist` *Transfer dissimilarity between phylogenetic trees*

Description

Compute the transfer dissimilarity between phylogenetic trees, as defined by Takazawa et al. (2026). The transfer dissimilarity uses the transfer distance (Lemoine et al. 2018) to compare bipartitions, providing a finer-grained measure than the Robinson–Foulds distance. Each split in each tree is scored by how many taxa must be moved to match its closest counterpart in the other tree, and these scores are summed.

Usage

```
TransferDist(
  tree1,
  tree2 = NULL,
  scale = TRUE,
  normalize = FALSE,
  reportMatching = FALSE
)

TransferDistance(
  tree1,
  tree2 = NULL,
  scale = TRUE,
  normalize = FALSE,
  reportMatching = FALSE
)

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

Arguments

`tree1`, `tree2`	Trees of class `phylo`, with leaves labelled identically, or lists of such trees to undergo pairwise comparison. Where implemented, `tree2 = NULL` will compute distances between each pair of trees in the list `tree1` using a fast algorithm based on Day (1985).
`scale`	Logical; if `TRUE` (default), use the scaled transfer dissimilarity. If `FALSE`, use the unscaled transfer dissimilarity.

`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`, `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 `scaled` variant divides each split's contribution by its depth minus one, giving equal weight to all splits regardless of their depth (analogous to the Robinson–Foulds distance). The `unscaled` variant uses raw transfer distances, giving more weight to deep splits. Neither variant satisfies the triangle inequality for trees with six or more tips.

Value

`TransferDist()` returns an object of class `dist` (if `tree2` is `NULL`), a numeric matrix (if both `tree1` and `tree2` are lists), or a numeric value (for a single pair). If `reportMatching = TRUE`, the return value carries `matching` and `pairScores` attributes.

Normalization

When `normalize = TRUE`, the scaled transfer dissimilarity is divided by `2 * (n - 3)`, placing it in the range [0, 1]. The unscaled version is divided by the maximum possible unscaled dissimilarity (following Takazawa et al. (2026)).

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

Lemoine F, Domelevo Entfellner J, Wilkinson E, Correia D, Dávila Felipe M, De Oliveira T, Gascuel O (2018). "Renewing Felsenstein's phylogenetic bootstrap in the era of big data." *Nature*, **556**(7702), 452–456. doi:10.1038/s41586-018-0043-0.

Takazawa Y, Takeda A, Hayamizu M, Gascuel O (2026). "Outperforming the majority-rule consensus tree using fine-grained dissimilarity measures." *bioRxiv*. doi:10.64898/2026.03.16.712085.

See Also

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

Examples

```
library(TreeTools)
TransferDist(BalancedTree(8), PectinateTree(8))
TransferDist(BalancedTree(8), PectinateTree(8), scale = FALSE)

# All-pairs
TransferDist(as.phylo(0:5, 8))
```

`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 = NULL)

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

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

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

ClusteringInfoDistance(
  tree1,
  tree2 = NULL,
  normalize = FALSE,
```

```
  reportMatching = FALSE
)

ExpectedVariation(tree1, tree2, samples = 10000)

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

MatchingSplitInfoDistance(
  tree1,
  tree2 = NULL,
  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. Where implemented, tree2 = NULL will compute distances between each pair of trees in the list tree1 using a fast algorithm based on Day (1985).
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. Where trees contain different tips, tips present in one tree but not the other are removed before each comparison (as by definition, the trees neither hold information in common nor differ regarding these tips).

Value

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

If reportMatching = TRUE, the functions additionally return an integer vector listing the index of the split in tree2 that is matched with each split in tree1 in the optimal matching. Unmatched splits are denoted NA. Use VisualizeMatching() to plot the optimal matching.

TreeDistance() simply returns the clustering information distance (it is an alias of ClusteringInfoDistance()).

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, use `normalize = pmax` or `pmin` to scale against the information content or entropy of all splits in the most (`pmax`) or least (`pmin`) informative tree in each pair. 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). For worked examples, see the internal function `NormalizeInfo()`, which is called from distance functions with the parameter `how = normalize`. .

Distances between large trees

To balance memory demands and runtime with flexibility, these functions are implemented for trees with up to 2048 leaves. To analyse trees with up to 8192 leaves, you will need to a modified version of the package: `install.packages("BigTreeDist", repos = "https://ms609.github.io/packages/")`

Use `library("BigTreeDist")` *instead* of `library("TreeDist")` to load the modified package – or prefix functions with the package name, e.g. `BigTreeDist::TreeDistance()`.

As an alternative download method, uninstall **TreeDist** and **TreeTools** using `remove.packages()`, then use `devtools::install_github("ms609/TreeTools", ref = "more-leaves")` to install the modified **TreeTools** package; then, install **TreeDist** using `devtools::install_github("ms609/TreeDist", ref = "more-leaves")`. (**TreeDist** will need building from source *after* the modified **TreeTools** package has been installed, as its code links to values set in the TreeTools source code.)

Trees with over 8192 leaves require further modification of the source code, which the maintainer plans to attempt in the future; please comment on GitHub if you would find this useful.

Parallelism

When `tree2 = NULL` and all trees share the same tip labels, pairwise distance calculation uses a multi-threaded OPENMP batch path automatically. Control the number of threads with the `"mc.cores"` option:

```
options(mc.cores = parallel::detectCores())  # use all cores
options(mc.cores = 4L)                        # or a fixed number
```

Do **not** call `StartParallel()` for these functions: a registered R cluster disables the OPENMP path and replaces it with slower inter-process communication. See `StartParallel()` for full guidance on when an R cluster is appropriate.

Author(s)

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

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28. doi:10.1007/BF01908061.

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

Meila 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*, **36**(20), 5007–5013. 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: `HierarchicalMutualInfo()`, `JaccardRobinsonFoulds()`, `KendallColijn()`, `MASTSize()`, `MatchingSplitDistance()`, `NNIDist()`, `NNIMaxStep()`, `NyeSimilarity()`, `PathDist()`, `Robinson-Foulds`, `SPRDist()`, `TransferDist()`

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
attr(SharedPhylogeneticInfo(tree1, tree2, reportMatching = TRUE), "matching")
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

# Distance functions internally convert trees to Splits objects.
# Pre-conversion can reduce run time if the same trees will feature in
# multiple comparisons
splits1 <- TreeTools::as.Splits(tree1)
splits2 <- TreeTools::as.Splits(tree2)

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

TreeInfo *Information content of splits within a tree*

Description

Sum the entropy (`ClusteringEntropy()`), clustering information content (`ClusteringInfo()`), or phylogenetic information content (`SplitwiseInfo()`) across each split within a phylogenetic tree, or the consensus of a set of phylogenetic trees (`ConsensusInfo()`). This value will be greater than the total information content of the tree where a tree contains multiple splits, as these splits are not independent and thus contain mutual information that is counted more than once

Usage

```
SplitwiseInfo(x, p = NULL, sum = TRUE)

ClusteringEntropy(x, p = NULL, sum = TRUE)

ClusteringInfo(x, p = NULL, sum = TRUE)

## S3 method for class 'phylo'
ClusteringEntropy(x, p = NULL, sum = TRUE)

## S3 method for class 'list'
ClusteringEntropy(x, p = NULL, sum = TRUE)

## S3 method for class 'multiPhylo'
ClusteringEntropy(x, p = NULL, sum = TRUE)

## S3 method for class 'Splits'
ClusteringEntropy(x, p = NULL, sum = TRUE)

## S3 method for class 'phylo'
ClusteringInfo(x, p = NULL, sum = TRUE)

## S3 method for class 'list'
ClusteringInfo(x, p = NULL, sum = TRUE)

## S3 method for class 'multiPhylo'
ClusteringInfo(x, p = NULL, sum = TRUE)

## S3 method for class 'Splits'
ClusteringInfo(x, p = NULL, sum = TRUE)

ConsensusInfo(trees, info = "phylogenetic", p = 0.5, check.tips = TRUE)
```

Arguments

x	A tree of class phylo, a list of trees, or a multiPhylo object.
p	Scalar from 0.5 to 1 specifying minimum proportion of trees that must contain a split for it to appear within the consensus.
sum	Logical: if TRUE, sum the information content of each split to provide the total splitwise information content of the tree.
trees	List of phylo objects, optionally with class multiPhylo.
info	Abbreviation of "phylogenetic" or "clustering", specifying the concept of information to employ.
check.tips	Logical specifying whether to renumber leaves such that leaf numbering is consistent in all trees.

Value

`SplitwiseInfo()`, `ClusteringInfo()` and `ClusteringEntropy()` return the splitwise information content of the tree – or of each split in turn, if `sum = FALSE` – in bits.

`ConsensusInfo()` returns the splitwise information content of the majority rule consensus of `trees`.

Clustering information

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 (Meila2007; Vinh et al. 2010).

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 (Vinh2010: p. 2840), the information content of a split is its entropy multiplied by the number of leaves.

Phylogenetic information

Phylogenetic information expresses the information content of a split in terms of the probability that a uniformly selected tree will contain it (Thorley et al. 1998).

Consensus information

The information content of splits in a consensus tree is calculated by interpreting support values (i.e. the proportion of trees containing each split in the consensus) as probabilities that the true tree contains that split, following Smith (2022).

Author(s)

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

References

Smith MR (2022). "Using information theory to detect rogue taxa and improve consensus trees." *Systematic Biology*, syab099. doi:10.1093/sysbio/syab099.

Thorley JL, Wilkinson M, Charleston M (1998). "The information content of consensus trees." In Rizzi A, Vichi M, Bock H (eds.), *Advances in Data Science and Classification*, 91–98. Springer, Berlin. doi:10.1007/9783642722530_12.

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

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

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

Examples

```
library("TreeTools", quietly = TRUE)

SplitwiseInfo(PectinateTree(8))
tree <- read.tree(text = "(a, b, (c, (d, e, (f, g)0.8))0.9);")
SplitwiseInfo(tree)
SplitwiseInfo(tree, TRUE)

# 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)

# Clustering entropy with uncertain splits
tree <- ape::read.tree(text = "(a, b, (c, (d, e, (f, g)0.8))0.9);")
ClusteringInfo(tree)
ClusteringInfo(tree, TRUE)

# Support-weighted information content of a consensus tree
set.seed(0)
trees <- list(RandomTree(8), RootTree(BalancedTree(8), 1), PectinateTree(8))
cons <- consensus(trees, p = 0.5)
p <- SplitFrequency(cons, trees) / length(trees)
plot(cons)
LabelSplits(cons, signif(SplitwiseInfo(cons, p, sum = FALSE), 4))
ConsensusInfo(trees)
LabelSplits(cons, signif(ClusteringInfo(cons, p, sum = FALSE), 4))
ConsensusInfo(trees, "clustering")
```

`VisualizeMatching` *Visualize 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.cex = par("cex"),
  value.cex = edge.cex * 0.8,
  edge.frame = "rect",
  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.cex`	Character expansion for edge labels. If `FALSE`, suppress edge labels.
`value.cex`	Character expansion for values on edge labels. If `FALSE`, values are not displayed.
`edge.frame`	Character specifying the kind of frame to be printed around the text of the edge labels. Choose an abbreviation of `"rect"`, `"circle"`, or `"none"`.
`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.

Value

`VisualizeMatching()` invisibly returns the matching of splits between `tree1` and `tree2` (i.e. `Func(tree1, tree2, reportMatching = TRUE)`)

Author(s)

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

Examples

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

VisualizeMatching(RobinsonFouldsMatching, tree1, tree2)
matching <- VisualizeMatching(SharedPhylogeneticInfo, tree1, tree2,
                              matchZeros = FALSE)
attributes(matching)
```

Index

∗ **cluster functions**
  cluster-statistics, 5
  KMeansPP, 18
∗ **information functions**
  SplitEntropy, 47
  SplitSharedInformation, 49
  TreeInfo, 63
∗ **pairwise tree distances**
  CompareAll, 6
∗ **summary trees**
  MCITree, 28
  TransferConsensus, 54
∗ **tree distances**
  HierarchicalMutualInfo, 8
  JaccardRobinsonFoulds, 13
  KendallColijn, 16
  MASTSize, 24
  MatchingSplitDistance, 26
  NNIDist, 33
  NNIMaxStep, 36
  NyeSimilarity, 37
  PathDist, 39
  Robinson-Foulds, 43
  SPRDist, 51
  TransferDist, 56
  TreeDistance, 58
∗ **tree space functions**
  cluster-statistics, 5
  Islands, 12
  MappingQuality, 21
  MapTrees, 22
  median.multiPhylo, 29
  MSTSegments, 32
  SpectralEigens, 46

AHMI *(*HierarchicalMutualInfo*)*, 8
AllSplitPairings, 4
ape::consensus(), *30*
as.HPart *(*HPart*)*, 11
as.phylo.HPart *(*HPart*)*, 11
cluster-statistics, 5
Clustering Information Distance, *36*
ClusteringEntropy *(*TreeInfo*)*, 63
ClusteringInfo *(*TreeInfo*)*, 63
ClusteringInfoDist *(*TreeDistance*)*, 58
ClusteringInfoDistance *(*TreeDistance*)*, 58
ClusteringInfoDistance(), *29*, *36*, *53*
CompareAll, 6
CompareAll(), *53*
ConsensusInfo *(*TreeInfo*)*, 63

DifferentPhylogeneticInfo *(*TreeDistance*)*, 58
DifferentPhylogeneticInfo(), *53*
DisplayMatching *(*VisualizeMatching*)*, 67
DistanceFromMedian *(*cluster-statistics*)*, 5
DistFromMed *(*cluster-statistics*)*, 5

EHMI *(*HierarchicalMutualInfo*)*, 8
Entropy, 7
ExpectedVariation *(*TreeDistance*)*, 58

GetParallel *(*StartParallel*)*, 52

HierarchicalMutualInfo, 8
HierarchicalMutualInfo(), *11*, *15*, *18*, *25*, *27*, *35*, *36*, *39*, *41*, *46*, *52*, *57*, *62*
HMI *(*HierarchicalMutualInfo*)*, 8
HPart, *9*, 11

InfoRobinsonFoulds *(*Robinson-Foulds*)*, 43
InfoRobinsonFoulds(), *53*
InfoRobinsonFouldsSplits *(*Robinson-Foulds*)*, 43
is.HPart *(*HPart*)*, 11
Islands, 12
Islands(), *6*, *22*, *24*, *30*, *33*, *47*

JaccardRobinsonFoulds, 13, *38*

JaccardRobinsonFoulds(), *10*, *18*, *25*, *27*, *35*, *36*, *39*, *41*, *46*, *52*, *53*, *57*, *62*
JaccardSplitSimilarity (JaccardRobinsonFoulds), 13

KCDiameter (KendallColijn), 16
KCVector (KendallColijn), 16
KendallColijn, 16
KendallColijn(), *10*, *15*, *25*, *27*, *35*, *36*, *39*, *41*, *46*, *52*, *57*, *62*
kmeans, *18*, *19*
KMeansPP, 18
KMeansPP(), *6*

LAPJV, 20
LnTreesConsistentWithTwoSplits (SplitSharedInformation), 49
Log2TreesConsistentWithTwoSplits (SplitSharedInformation), 49

mad, *5*
makeCluster(), *53*
MappingQuality, 21
MappingQuality(), *6*, *13*, *24*, *30*, *33*, *47*
MapTrees, 22
MapTrees(), *6*, *13*, *22*, *30*, *33*, *47*
MASTInfo (MASTSize), 24
MASTInfo(), *53*
MASTSize, 24
MASTSize(), *10*, *15*, *18*, *27*, *35*, *36*, *39*, *41*, *46*, *52*, *53*, *57*, *62*
MatchingSplitDistance, 26
MatchingSplitDistance(), *10*, *15*, *18*, *25*, *35*, *36*, *39*, *41*, *46*, *52*, *53*, *57*, *62*
MatchingSplitDistanceSplits (MatchingSplitDistance), 26
MatchingSplitInfo (TreeDistance), 58
MatchingSplitInfo(), *53*
MatchingSplitInfoDistance (TreeDistance), 58
MatchingSplitInfoDistance(), *53*
MatchingSplitInfoSplits (TreeDistance), 58
MCITree, 28
MCITree(), *55*
MeanCentDist (cluster-statistics), 5
MeanCentroidDist (cluster-statistics), 5
MeanCentroidDistance (cluster-statistics), 5
MeanMSTEdge (cluster-statistics), 5
MeanNN (cluster-statistics), 5
median.multiPhylo, 29
median.multiPhylo(), *6*, *13*, *22*, *24*, *33*, *47*
MeilaMutualInformation (MeilaVariationOfInformation), 30
MeilaVariationOfInformation, 30
MSTSegments, 32
MSTSegments(), *6*, *13*, *22*, *24*, *30*, *47*
MutualClusteringInfo (TreeDistance), 58
MutualClusteringInfo(), *53*
MutualClusteringInformation (TreeDistance), 58
MutualClusteringInfoSplits (TreeDistance), 58

NNIDiameter (NNIDist), 33
NNIDiameter(), *36*
NNIDist, 33
NNIDist(), *10*, *15*, *18*, *25*, *27*, *36*, *39*, *41*, *46*, *52*, *53*, *57*, *62*
NNIMaxStep, 36
NNIMaxStep(), *10*, *15*, *18*, *25*, *27*, *35*, *39*, *41*, *46*, *52*, *57*, *62*
NormalizeInfo(), *61*
Ntropy (Entropy), 7
NyeSimilarity, 37
NyeSimilarity(), *10*, *15*, *18*, *25*, *27*, *35*, *36*, *41*, *46*, *52*, *53*, *57*, *62*
NyeSplitSimilarity (NyeSimilarity), 37

path.dist(), *7*, *40*
PathDist, 39
PathDist(), *10*, *15*, *18*, *25*, *27*, *35*, *36*, *39*, *46*, *52*, *57*, *62*
PathVector (KendallColijn), 16
phangorn::mast(), *25*
phylo, *16*
PhylogeneticInfoDistance (TreeDistance), 58
plot.default(), *42*
plot.HPart (HPart), 11
plot.phylo, *11*
Plot3, 41
PlotMatching (VisualizeMatching), 67
pmax, *61*
pmin, *61*
print.HPart (HPart), 11

Project *(MapTrees)*, 22
ProjectionQuality *(MappingQuality)*, 21

ReduceTrees, 42
Robinson-Foulds, 43
RobinsonFoulds *(Robinson-Foulds)*, 43
RobinsonFouldsInfo *(Robinson-Foulds)*, 43
RobinsonFouldsMatching *(Robinson-Foulds)*, 43
RobinsonFouldsSplits *(Robinson-Foulds)*, 43

segments(), *32*
SelfHMI *(HierarchicalMutualInfo)*, 8
SetParallel *(StartParallel)*, 52
SharedPhylogeneticInfo *(TreeDistance)*, 58
SharedPhylogeneticInfo(), *53*
SharedPhylogeneticInfoSplits *(TreeDistance)*, 58
SpectralClustering *(SpectralEigens)*, 46
SpectralEigens, 46
SpectralEigens(), *6*, *13*, *22*, *24*, *30*, *33*
SplitDifferentInformation *(SplitSharedInformation)*, 49
SplitEntropy, 47
SplitEntropy(), *50*, *66*
SplitsCompatible, 48
SplitSharedInformation, 49
SplitSharedInformation(), *48*, *66*
SplitVector *(KendallColijn)*, 16
SplitwiseInfo *(TreeInfo)*, 63
SplitwiseInfo(), *65*
SPR.dist(), *52*
SPRDist, 51
SPRDist(), *10*, *15*, *18*, *25*, *27*, *35*, *36*, *39*, *41*, *46*, *57*, *62*
StartParallel, 52
StartParallel(), *62*
StopParallel *(StartParallel)*, 52
StrainCol *(MSTSegments)*, 32
SumOfRanges *(cluster-statistics)*, 5
SumOfVariances *(cluster-statistics)*, 5
SumOfVars *(cluster-statistics)*, 5

TransferConsensus, 54
TransferConsensus(), *28*
TransferDist, 56
TransferDist(), *10*, *15*, *18*, *25*, *27*, *35*, *36*, *39*, *41*, *46*, *52*, *62*
TransferDistance *(TransferDist)*, 56
TransferDistSplits *(TransferDist)*, 56
TreeDistance, 58
TreeDistance(), *10*, *15*, *18*, *25*, *27*, *35*, *36*, *39*, *41*, *43*, *46*, *52*, *57*
TreeInfo, *48*, *50*, 63
TreesConsistentWithTwoSplits *(SplitSharedInformation)*, 49
TreeTools::ConsensusWithout(), *30*
TreeTools::MSTEdges(), *32*

vapply(), *7*
VisualiseMatching *(VisualizeMatching)*, 67
VisualizeMatching, 67
VisualizeMatching(), *45*, *46*, *60*