Package: TreeSearch (via r-universe)

October 23, 2026

Title Phylogenetic Analysis with Discrete Character Data

Version 1.8.0.9001

License GPL (>= 3)

Copyright Incorporates C/C++ code from Morphy Phylogenetic Library by Martin Brazeau <https://github.com/mbrazeau/MorphyLib> (GPL3)

Description Reconstruct phylogenetic trees from discrete data. Inapplicable character states are handled using the algorithm of Brazeau, Guillerme and Smith (2019) <doi:10.1093/sysbio/syy083> with the ``Morphy'' library, under equal or implied step weights. Contains a ``shiny'' user interface for interactive tree search and exploration of results, including character visualization, rogue taxon detection, tree space mapping, and cluster consensus trees (Smith 2022a, b) <doi:10.1093/sysbio/syab099>, <doi:10.1093/sysbio/syab100>. Profile Parsimony (Faith and Trueman, 2001) <doi:10.1080/10635150118627>, Successive Approximations (Farris, 1969) <doi:10.2307/2412182> and custom optimality criteria are implemented.

URL https://ms609.github.io/TreeSearch/ (doc), https://github.com/ms609/TreeSearch/ (devel)

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

Depends R (>= 4.0)

Imports abind, ape (>= 5.6), base64enc, cli (>= 3.0), cluster, colorspace, fastmap, fastmatch (>= 1.1.3), fs, future, PlotTools, promises, protoclust, Rcpp, Rdpack (>= 0.7), Rogue (> 2.0.0), shiny (>= 1.6.0), shinyjs, stats, stringi, TreeDist (>= 2.6.3), TreeTools (>= 1.16),

Suggests knitr, phangorn (>= 2.2.1), Quartet, readxl, rmarkdown, shinytest, spelling, testthat, vdiffr (>= 1.0.0),

Config/Needs/check callr, pkgbuild, rcmdcheck,

Config/Needs/coverage covr, spelling

Config/Needs/memcheck devtools, remotes

Config/Needs/metadata codemeta

Config/Needs/revdeps revdepcheck

Config/Needs/website curl, igraph, pkgdown,

Config/roxygen2/version 8.0.0

RdMacros Rdpack

LinkingTo Rcpp, TreeTools,

SystemRequirements C++17

LazyData true

ByteCompile true

Encoding UTF-8

Language en-GB

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

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

Date/Publication 2026-07-30 07:36:04 UTC

RemoteUrl https://github.com/ms609/TreeSearch

RemoteRef HEAD

RemoteSha 78ee3d969fb7ab79eb054789dc4ec37da331ba3e

Contents

AdditionTree . . . 3
AllSPR . . . 4
Carter1 . . . 5
CharacterLength . . . 6
ClusterStrings . . . 8
ConcordanceTable . . . 9
ConcordantInformation . . . 11
congreveLamsdellMatrices . . . 12
Consistency . . . 13
cSPR . . . 15
ExpectedLength . . . 15
GapHandler . . . 16
inapplicable.datasets . . . 17
is.morphyPtr . . . 21
IWScore . . . 21
Jackknife . . . 24
JackLabels . . . 25
LengthAdded . . . 27
MaximizeParsimony . . . 29
MinimumLength . . . 36

MorphyBootstrap . . . 38
MorphyWeights . . . 41
MostContradictedFreq . . . 43
NNI . . . 44
PhyDat2Morphy . . . 46
PlotCharacter . . . 47
PrepareDataProfile . . . 50
PresCont . . . 51
profiles . . . 53
QACol . . . 54
QuartetResolution . . . 55
RandomMorphyTree . . . 56
RandomTreeScore . . . 56
RearrangeEdges . . . 57
referenceTree . . . 59
SingleCharMorphy . . . 59
SiteConcordance . . . 60
SPR . . . 64
StepInformation . . . 67
summary.morphyPtr . . . 68
TaxonInfluence . . . 68
TBR . . . 71
UnloadMorphy . . . 73
WhenFirstHit . . . 74
WithOneExtraStep . . . 75

Index **76**

`AdditionTree` *Addition tree*

Description

Generates a starting tree by adding each taxon in turn to the most parsimonious location.

Usage

```
AdditionTree(dataset, concavity = Inf, constraint, sequence)
```

Arguments

`dataset`	A phylogenetic data matrix of **phangorn** class `phyDat`, whose names correspond to the labels of any accompanying tree. Perhaps load into R using `ReadAsPhyDat()`. Additive (ordered) characters can be handled using `Decompose()`.

concavity	Determines the degree to which extra steps beyond the first are penalized. Specify a numeric value to use implied weighting (Goloboff 1993); concavity specifies k in $k / e + k$. A value of 10 is recommended; TNT sets a default of 3, but this is too low in some circumstances (Goloboff et al. 2018; Smith 2019). Better still explore the sensitivity of results under a range of concavity values, e.g. k = 2 ^ (1:7). Specify Inf to weight each additional step equally, (which underperforms step weighting approaches (Goloboff et al. 2008; Goloboff et al. 2018; Goloboff and Arias 2019; Smith 2019)). Specify "profile" to employ an approximation of profile parsimony (Faith and Trueman 2001).
constraint	Either an object of class phyDat, in which case returned trees will be perfectly compatible with each character in constraint; or a tree of class phylo, all of whose nodes will occur in any output tree. See ImposeConstraint() and vignette for further examples.
sequence	Character or numeric vector listing sequence in which to add taxa. Randomized if not provided.

Value

AdditionTree() returns a tree of class phylo, rooted on sequence[1].

Author(s)

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

See Also

Impose a constraint: TreeTools::ImposeConstraint()

Neighbour-joining trees: TreeTools::NJTree(); TreeTools::ConstrainedNJ()

Other tree generation functions: RandomMorphyTree()

Examples

```
data("inapplicable.phyData", package = "TreeSearch")
AdditionTree(inapplicable.phyData[["Longrich2010"]], concavity = 10)
```

AllSPR *All SPR trees*

Description

All SPR trees

Usage

```
AllSPR(parent, child, nEdge, notDuplicateRoot, edgeToBreak)
```

Arguments

parent	Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 1]
child	Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 2].
nEdge	integer specifying the number of edges of a tree of class phylo, i.e. dim(tree$edge)[1]
notDuplicateRoot	logical vector of length nEdge, specifying for each edge whether it is the second edge leading to the root (in which case its breaking will be equivalent to breaking the other root edge... except insofar as it moves the position of the root.)
edgeToBreak	(optional) integer specifying the index of an edge to bisect/prune, generated randomly if not specified. Alternatively, set to -1 to return a complete list of all trees one step from the input tree.

Value

AllSPR() returns a list of edge matrices for all trees one SPR rearrangement from the starting tree

Author(s)

Martin R. Smith

Carter1 *Number of trees with* m *steps*

Description

Calculate the number of trees in which Fitch parsimony will reconstruct *m* steps, where *a* leaves are labelled with one state, and *b* leaves are labelled with a second state.

Usage

```
Carter1(m, a, b)

Log2Carter1(m, a, b)

LogCarter1(m, a, b)
```

Arguments

m	Number of steps.
a, b	Number of leaves labelled 0 and 1.

Details

Implementation of theorem 1 from Carter et al. (1990)

Author(s)

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

References

Carter M, Hendy M, Penny D, Székely LA, Wormald NC (1990). "On the distribution of lengths of evolutionary trees." *SIAM Journal on Discrete Mathematics*, **3**(1), 38–47. doi:10.1137/0403005.

See also:

Steel MA (1993). "Distributions on bicoloured binary trees arising from the principle of parsimony." *Discrete Applied Mathematics*, **41**(3), 245–261. doi:10.1016/0166218X(90)90058K.

Steel M, Charleston M (1995). "Five surprising properties of parsimoniously colored trees." *Bulletin of Mathematical Biology*, **57**(2), 367–375. doi:10.1016/00928240(94)00051D.

(Steel M, Goldstein L, Waterman MS (1996). "A central limit theorem for the parsimony length of trees." *Advances in Applied Probability*, **28**(4), 1051–1071. doi:10.2307/1428164.)

See Also

Other profile parsimony functions: `PrepareDataProfile()`, `StepInformation()`, `WithOneExtraStep()`, `profiles`

Examples

```
# The character `0 0 0 1 1 1`
Carter1(1, 3, 3) # Exactly one step
Carter1(2, 3, 3) # Two steps (one extra step)

# Number of trees that the character can map onto with exactly _m_ steps
# if non-parsimonious reconstructions are permitted:
cumsum(sapply(1:3, Carter1, 3, 3))

# Three steps allow the character to map onto any of the 105 six-leaf trees.
```

`CharacterLength` *Character length*

Description

Homoplasy length of each character in a dataset on a specified tree.

Usage

```
CharacterLength(tree, dataset, compress = FALSE)

FastCharacterLength(tree, dataset)
```

Arguments

tree	A tree of class phylo.
dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
compress	Logical specifying whether to retain the compression of a phyDat object or to return a vector specifying to each individual character, decompressed using the dataset's index attribute.

Value

CharacterLength() returns a vector listing the contribution of each character to tree score, according to the algorithm of Brazeau et al. (2019).

Functions

- FastCharacterLength(): Do not perform checks. Use with care: may cause erroneous results or software crash if variables are in the incorrect format.

Author(s)

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

References

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

See Also

Other tree scoring: ExpectedLength(), IWScore(), LengthAdded(), MinimumLength(), MorphyTreeLength(), TaxonInfluence()

Examples

```
data("inapplicable.datasets")
dataset <- inapplicable.phyData[[12]]
tree <- TreeTools::NJTree(dataset)
CharacterLength(tree, dataset)
CharacterLength(tree, dataset, compress = TRUE)
```

`ClusterStrings` *Cluster similar strings*

Description

Calculate string similarity using the Levenshtein distance and return clusters of similar strings.

Usage

```
ClusterStrings(x, maxCluster = 12)
```

Arguments

`x`	Character vector.
`maxCluster`	Integer specifying maximum number of clusters to consider.

Value

`NameClusters()` returns an integer assigning each element of `x` to a cluster, with an attribute `med` specifying the median string in each cluster, and `silhouette` reporting the silhouette coefficient of the optimal clustering. Coefficients < 0.5 indicate weak structure, and no clusters are returned. If the number of unique elements of `x` is less than `maxCluster`, all occurrences of each entry are assigned to an individual cluster.

Author(s)

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

See Also

Other utility functions: `QACol()`, `QuartetResolution()`, `WhenFirstHit()`

Examples

```
ClusterStrings(c(paste0("FirstCluster ", 1:5),
                 paste0("SecondCluster.", 8:12),
                 paste0("AnotherCluster_", letters[1:6])))
```

`ConcordanceTable` *Plot concordance table*

Description

`ConcordanceTable()` plots a concordance table (Smith 2026).

Usage

```
ConcordanceTable(
  tree,
  dataset,
  Col = QACol,
  largeClade = 0,
  xlab = "Edge",
  ylab = "Character",
  normalize = TRUE,
  plot = TRUE,
  marginSize = 0L,
  ...
)
```

Arguments

`tree`	A tree of class `phylo`.
`dataset`	A phylogenetic data matrix of **phangorn** class `phyDat`, whose names correspond to the labels of any accompanying tree. Perhaps load into R using `ReadAsPhyDat()`. Additive (ordered) characters can be handled using `Decompose()`.
`Col`	Function that takes vectors `amount` and `quality` and returns a vector of colours. QCol colours by data quality (concordance); QACol by quality and amount of information.
`largeClade`	Integer; if greater than 1, vertical lines will be drawn at edges whose descendants are both contain more than `largeClade` leaves.
`xlab`	Character giving a label for the x axis.
`ylab`	Character giving a label for the y axis.
`normalize`	Controls how the *expected* mutual information (the zero point of the scale) is determined. • `FALSE`: no chance correction; MI is scaled only by its maximum. • `TRUE`: subtract the analytical expected MI for random association. • `<integer>`: subtract an empirical expected MI estimated from that number of random trees. In all cases, 1 corresponds to the maximal attainable MI for the pair (`hBest`), and 0 corresponds to the chosen expectation.
`plot`	Logical specifying whether to draw the plot.

`marginSize` Integer scalar or vector controlling summary margin strips. If a scalar (length 1) and greater than zero, both a left strip and a bottom strip are added, each `marginSize` grid cells wide/tall. If a vector (length > 1), each entry controls one side following the usual `par(mar)` order — `c(bottom, left, top, right)` — where a positive value enables that strip with the given width/height and `NA` or `0` suppresses it. Currently only the bottom (entry 1) and left (entry 2) strips are implemented; further entries are accepted but ignored. The left strip is coloured by the characterwise concordance (weighted mean across edges); the bottom strip by the edgewise concordance (weighted mean across characters). One blank cell separates each strip from the main grid.

`...` Arguments to `abline`, to control the appearance of vertical lines marking important edges.

Value

`ConcordanceTable()` invisibly returns an named list containing:

- `"info"`: The amount of information in each character-edge pair, in bits;
- `"relInfo"`: The information, normalized to the most information-rich pair;
- `"quality"`: The normalized mutual information of the pair;
- `"col"`: The colours used to plot the table.

References

Smith MR (2026). "Which characters support which clades? Exploring the distribution of phylogenetic signal using mutual information." *Systematic Biology*, Under review.

See Also

- `SiteConcordance()`: compute underlying concordance values.

Other split support functions: `JackLabels()`, `Jackknife()`, `MaximizeParsimony()`, `MostContradictedFreq()`, `PresCont()`, `SiteConcordance`

Examples

```
# Load data and tree
data("congreveLamsdellMatrices", package = "TreeSearch")
dataset <- congreveLamsdellMatrices[[1]][, 1:20]
tree <- referenceTree

# Plot tree and identify nodes
library("TreeTools", quietly = TRUE)
plot(tree)
nodeIndex <- as.integer(rownames(as.Splits(tree)))
nodelabels(seq_along(nodeIndex), nodeIndex, adj = c(2, 1),
           frame = "none", bg = NULL)
QALegend(where = c(0.1, 0.4, 0.1, 0.3))

# View information shared by characters and edges
```

```
ConcordanceTable(tree, dataset, largeClade = 3, col = 2, lwd = 3,
                 marginSize = 1:4)
axis(1)
axis(2)

# Visualize dataset
image(t(`mode<-`(PhyDatToMatrix(dataset), "numeric")), axes = FALSE,
      xlab = "Leaf", ylab = "Character")
```

`ConcordantInformation` *Evaluate the concordance of information between a tree and a dataset*

Description

Details the amount of information in a phylogenetic dataset that is consistent with a specified phylogenetic tree, and the signal:noise ratio of the character matrix implied if the tree is true.

Usage

```
ConcordantInformation(tree, dataset)

Evaluate(tree, dataset)

ConcordantInfo(tree, dataset)
```

Arguments

`tree`	A tree of class `phylo`.
`dataset`	A phylogenetic data matrix of **phangorn** class `phyDat`, whose names correspond to the labels of any accompanying tree. Perhaps load into R using `ReadAsPhyDat()`. Additive (ordered) characters can be handled using `Decompose()`.

Details

Presently restricted to datasets whose characters contain a maximum of two parsimony-informative states.

Value

`ConcordantInformation()` returns a named vector with elements:

- `informationContent`: cladistic information content of `dataset`
- `signal`, `noise`: amount of cladistic information that represents phylogenetic signal and noise, according to `tree`
- `signalToNoise`: the implied signal:noise ratio of `dataset`
- `treeInformation`: the cladistic information content of a bifurcating tree on `dataset`; this is the minimum amount of information necessary to resolve a bifurcating tree, assuming no duplicate information or noise

- `matrixToTree`: the ratio of the cladistic information content of the matrix to the cladistic information content of the tree, a measure of the redundancy of the matrix
- `ignored`: information content of characters whose signal and noise could not be calculated (too many states) and so are not included in the totals above.

Author(s)

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

Examples

```
data(congreveLamsdellMatrices)
myMatrix <- congreveLamsdellMatrices[[10]]
ConcordantInformation(TreeTools::NJTree(myMatrix), myMatrix)
```

`congreveLamsdellMatrices`

100 simulated data matrices

Description

Contains the 100 simulated matrices generated by (Congreve and Lamsdell 2016) using a heterogeneous Markov-k model, generated from the referenceTree topology, with all branches sharing an equal length.

Usage

```
congreveLamsdellMatrices
```

Format

A list with 100 entries, each comprising a `phyDat` object of 55 characters for 22 taxa

Source

doi:10.5061/dryad.7dq0j

References

Congreve CR, Lamsdell JC (2016). "Implied weighting and its utility in palaeontological datasets: a study using modelled phylogenetic matrices." *Palaeontology*, **59**(3), 447–465. doi:10.1111/pala.12236.

Examples

```
data("referenceTree")
data("congreveLamsdellMatrices")
TreeLength(referenceTree, congreveLamsdellMatrices[[17]], "profile")
```

`Consistency` *Consistency and retention "indices"*

Description

`Consistency()` calculates the consistency "index" and retention index (Farris 1989) for each character in a dataset, given a bifurcating tree. Although there is not a straightforward interpretation of these indices, they are sometimes taken as an indicator of the fit of a character to a tree. Values correlate with the number of species sampled and the distribution of taxa between character states, so are not strictly comparable between characters in which these factors differ; and values cannot be compared between datasets (Speed and Arbuckle 2017).

Usage

```
Consistency(dataset, tree, nRelabel = 0, compress = FALSE)
```

Arguments

`dataset`	A phylogenetic data matrix of **phangorn** class `phyDat`, whose names correspond to the labels of any accompanying tree. Perhaps load into R using `ReadAsPhyDat()`. Additive (ordered) characters can be handled using `Decompose()`.
`tree`	A tree of class `phylo`.
`nRelabel`	Integer specifying how many times to relabel leaves when computing MCMC estimate of null tree length for RHI calculation. Steell et al. (2025) recommend 1000, but suggest that 100 may suffice. If zero (the default), the RHI is not calculated.
`compress`	Logical specifying whether to retain the compression of a `phyDat` object or to return a vector specifying to each individual character, decompressed using the dataset's `index` attribute.

Details

The **consistency "index"** (Kluge and Farris 1969) is defined as the number of steps observed in the most parsimonious mapping of a character to a tree, divided by the number of steps observed on the shortest possible tree for that character. A value of one indicates that a character's fit to the tree is optimal. Note that as the possible values of the consistency index do not range from zero to one, it is not an index in the mathematical sense of the term. Shortcomings of this measure are widely documented (Archie 1989-09; Brooks et al. 1986; Steell et al. 2025).

The maximum length of a character (see `MaximumLength()`) is the number of steps in a parsimonious reconstruction on the longest possible tree for a character. The **retention index** is the maximum length of a character minus the number of steps observed on a given tree; divided by the maximum length minus the minimum length. It is interpreted as the ratio between the observed homoplasy, and the maximum observed homoplasy, and scales from zero (worst fit that can be reconstructed under parsimony) to one (perfect fit).

The **rescaled consistency index** is the product of the consistency and retention indices; it rescales the consistency index such that its range of possible values runs from zero (least consistent) to one (perfectly consistent).

The **relative homoplasy index** (Steell et al. 2025) is the ratio of the observed excess tree length to the excess tree length due to chance, taken as the median score of a character when the leaves of the given tree are randomly shuffled.

The lengths of characters including inapplicable tokens are calculated following Brazeau et al. (2019), matching their default treatment in `TreeLength()`.

Value

`Consistency()` returns a matrix with named columns specifying the consistency index (`ci`), retention index (`ri`), rescaled consistency index (`rc`) and relative homoplasy index (`rhi`).

Author(s)

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

References

Archie JW (1989-09). "Homoplasy Excess Ratios: New Indices for Measuring Levels of Homoplasy in Phylogenetic Systematics and a Critique of the Consistency Index." *Systematic Zoology*, **38**(3), 253. doi:10.2307/2992286.

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

Brooks DR, O'Grady RT, Wiley EO (1986). "A Measure of the Information Content of Phylogenetic Trees, and Its Use as an Optimality Criterion." *Systematic Biology*, **35**(4), 571–581. doi:10.2307/2413116.

Farris JS (1989). "The Retention Index and the Rescaled Consistency Index." *Cladistics*, **5**(4), 417–419. doi:10.1111/j.10960031.1989.tb00573.x.

Kluge AG, Farris JS (1969). "Quantitative Phyletics and the Evolution of Anurans." *Systematic Zoology*, **18**(1), 1–32. doi:10.1093/sysbio/18.1.1.

Speed MP, Arbuckle K (2017). "Quantification Provides a Conceptual Basis for Convergent Evolution." *Biological Reviews*, **92**(2), 815–829. doi:10.1111/brv.12257.

Steell EM, Hsiang AY, Field DJ (2025). "Revealing Patterns of Homoplasy in Discrete Phylogenetic Datasets with a Cross-Comparable Index." *Zoological Journal of the Linnean Society*, **204**(1), zlaf024. doi:10.1093/zoolinnean/zlaf024.

Examples

```
data(inapplicable.datasets)
dataset <- inapplicable.phyData[[4]]
head(Consistency(dataset, TreeTools::NJTree(dataset), nRelabel = 10))
```

cSPR — cSPR() *expects a tree rooted on a single tip.*

Description

cSPR() expects a tree rooted on a single tip.

Usage

```
cSPR(tree, whichMove = NULL)
```

Arguments

tree	A tree of class phylo.
whichMove	Integer specifying which SPR move index to perform.

Author(s)

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

Examples

```
tree <- TreeTools::BalancedTree(8)

# Tree must be rooted on leaf
tree <- TreeTools::RootTree(tree, 1)

# Random rearrangement
cSPR(tree)

# Specific rearrangement
cSPR(tree, 9)
```

ExpectedLength — *Expected length*

Description

For a given dataset and tree topology, ExpectedLength() estimates the length expected if the states of each character are shuffled randomly across the leaves.

Usage

```
ExpectedLength(dataset, tree, nRelabel = 1000, compress = FALSE)
```

Arguments

dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
tree	A tree of class phylo.
nRelabel	Integer specifying how many times to relabel leaves when computing MCMC estimate of null tree length for RHI calculation. Steell et al. (2025) recommend 1000, but suggest that 100 may suffice. If zero (the default), the RHI is not calculated.
compress	Logical specifying whether to retain the compression of a phyDat object or to return a vector specifying to each individual character, decompressed using the dataset's index attribute.

Value

ExpectedLength() returns a numeric vector stating the median length of each character in dataset on tree after nRelabel random relabelling of leaves.

Author(s)

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

References

Steell EM, Hsiang AY, Field DJ (2025). "Revealing Patterns of Homoplasy in Discrete Phylogenetic Datasets with a Cross-Comparable Index." *Zoological Journal of the Linnean Society*, **204**(1), zlaf024. doi:10.1093/zoolinnean/zlaf024.

See Also

Other tree scoring: CharacterLength(), IWScore(), LengthAdded(), MinimumLength(), MorphyTreeLength(), TaxonInfluence()

GapHandler *Read how a Morphy Object handles the inapplicable token*

Description

Gaps represented by the inapplicable token can be treated as "missing data", i.e. as equivalent to the ambiguous token ?; as an extra state, equivalent to other states such as 0 or 1; or as "inapplicable data" using the algorithm of Brazeau, Guillerme and Smith (2019).

Usage

```
GapHandler(morphyObj)
```

Arguments

morphyObj	Object of class morphy, perhaps created with PhyDat2Morphy().

Value

GapHandler() returns a character string stating how gaps are handled by morphyObj.

Author(s)

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

See Also

Other Morphy API functions: MorphyErrorCheck(), MorphyWeights(), PhyDat2Morphy(), SingleCharMorphy(), UnloadMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

Examples

```
morphyObj <- SingleCharMorphy("-0-0", "Extra")
GapHandler(morphyObj)
morphyObj <- UnloadMorphy(morphyObj)
```

---

inapplicable.datasets *Thirty datasets with inapplicable data*

Description

These are the datasets used to evaluate the behaviour of the inapplicable algorithm in Brazeau et al. (2019). The name of each item corresponds to the datasets listed below. Datasets are sorted into two subsets, each sorted alphabetically; the first subset comprise simpler datasets with faster processing times. inapplicable.datasets provide the data in the matrix format generated by read.nexus.data(); inapplicable.phyData are in phyDat format. inapplicable.trees lists for each dataset a sample of up to 50 trees obtained by tree search under each inapplicable treatment, named accordingly. inapplicable.citations is a named character vector specifying the source of each dataset.

Usage

```
inapplicable.datasets

inapplicable.phyData
```

`inapplicable.trees`

`inapplicable.citations`

Format

An object of class `list` of length 30.

An object of class `list` of length 30.

An object of class `list` of length 31.

An object of class `character` of length 30.

Source

Subset one (faster processing):

Agnarsson2004 AGNARSSON, I. 2004. Morphological phylogeny of cobweb spiders and their relatives (Araneae, Araneoidea, Theridiidae). Zoological Journal of the Linnean Society, 141, 447–626.

Capa2011 CAPA, M., HUTCHINGS, P., AGUADO, M. T. and BOTT, N. J. 2011. Phylogeny of Sabellidae (Annelida) and relationships with other taxa inferred from morphology and multiple genes. Cladistics, 27, 449–469.

DeAssis2011 DE ASSIS, J. E. and CHRISTOFFERSEN, M. L. 2011. Phylogenetic relationships within Maldanidae (Capitellida, Annelida), based on morphological characters. Systematics and Biodiversity, 9, 233–245.

OLeary1999 O'LEARY, M. A. and GEISLER, J. H. 1999. The position of Cetacea within Mammalia: phylogenetic analysis of morphological data from extinct and extant taxa. Systematic Biology, 48, 455–490.

Rousset2004 ROUSSET, V., ROUSE, G. W., SIDDALL, M. E., TILLIER, A. and PLEIJEL, F. 2004. The phylogenetic position of Siboglinidae (Annelida) inferred from 18S rRNA, 28S rRNA and morphological data. Cladistics, 20, 518–533.

Sano2011 SANO, M. and AKIMOTO, S.-I. 2011. Morphological phylogeny of gall-forming aphids of the tribe Eriosomatini (Aphididae: Eriosomatinae). Systematic Entomology, 36, 607–627.

Sansom2010 SANSOM, R. S., FREEDMAN, K., GABBOTT, S. E., ALDRIDGE, R. J. and PURNELL, M. A. 2010. Taphonomy and affinity of an enigmatic Silurian vertebrate, *Jamoytius kerwoodi* White. Palaeontology, 53, 1393–1409.

Schulze2007 SCHULZE, A., CUTLER, E. B. and GIRIBET, G. 2007. Phylogeny of sipunculan worms: A combined analysis of four gene regions and morphology. Molecular Phylogenetics and Evolution, 42, 171–92.

Shultz2007 SHULTZ, J. W. 2007. A phylogenetic analysis of the arachnid orders based on morphological characters. Zoological Journal of the Linnean Society, 150, 221–265.

Wetterer2000 WETTERER, A. L., ROCKKMAN, M. V. and SIMMONS, N. B. 2000. Phylogeny of phyllostomid bats (Mammalia: Chiroptera): data from diverse morphological systems, sex chromosomes, and restriction sites. Bulletin of the American Museum of Natural History, 248, 1–200.

Wills2012 WILLS, M. A., GERBER, S., RUTA, M. and HUGHES, M. 2012. The disparity of priapulid, archaeopriapulid and palaeoscolecid worms in the light of new data. Journal of Evolutionary Biology, 25, 2056–2076.

Subset two (longer processing times):

Aguado2009 AGUADO, M. T. and SAN MARTIN, G. 2009. Phylogeny of Syllidae (Polychaeta) based on morphological data. Zoologica Scripta, 38, 379–402.

Aria2015 ARIA, C., CARON, J. B. and GAINES, R. 2015. A large new leanchoiliid from the Burgess Shale and the influence of inapplicable states on stem arthropod phylogeny. Palaeontology, 58, 629–660.

Asher2005 ASHER, R. J. and HOFREITER, M. 2006. Tenrec phylogeny and the noninvasive extraction of nuclear DNA. Systematic biology, 55, 181–94.

Baker2009 BAKER, W. J., SAVOLAINEN, V., ASMUSSEN-LANGE, C. B., CHASE, M. W., DRANSFIELD, J., FOREST, F., HARLEY, M. M., UHL, N. W. and WILKINSON, M. 2009. Complete generic-level phylogenetic analyses of palms (Arecaceae) with comparisons of supertree and supermatrix approaches. Systematic Biology, 58, 240–256.

Bouchenak2010 BOUCHENAK-KHELLADI, Y., VERBOOM, G. A., SAVOLAINEN, V. and HODKINSON, T. R. 2010. Biogeography of the grasses (Poaceae): a phylogenetic approach to reveal evolutionary history in geographical space and geological time. Botanical Journal of the Linnean Society, 162, 543–557.

Conrad2008 CONRAD, J. L. 2008. Phylogeny And Systematics Of Squamata (Reptilia) Based On Morphology. Bulletin of the American Museum of Natural History, 310, 1–182.

Dikow2009 DIKOW, T. 2009. A phylogenetic hypothesis for Asilidae based on a total evidence analysis of morphological and DNA sequence data (Insecta: Diptera: Brachycera: Asiloidea). Organisms Diversity and Evolution, 9, 165–188.

Eklund2004 EKLUND, H., DOYLE, J. A. and HERENDEEN, P. S. 2004. Morphological phylogenetic analysis of living and fossil Chloranthaceae. International Journal of Plant Sciences, 165, 107–151.

Geisler2001 GEISLER, J. H. 2001. New morphological evidence for the phylogeny of Artiodactyla, Cetacea, and Mesonychidae. American Museum Novitates, 3344, 53.

Giles2015 GILES, S., FRIEDMAN, M. and BRAZEAU, M. D. 2015. Osteichthyan-like cranial conditions in an Early Devonian stem gnathostome. Nature, 520, 82–85.

Griswold1999 GRISWOLD, C. E., CODDINGTON, J. A., PLATNICK, N. I. and FORSTER, R. R. 1999. Towards a phylogeny of entelegyne spiders (Araneae, Araneomorphae, Entelegynae). Journal of Arachnology, 27, 53–63.

Liljeblad2008 LILJEBLAD, J., RONQUIST, F., NIEVES-ALDREY, J. L., FONTAL-CAZALLA, F., ROS-FARRE, P., GAITROS, D. and PUJADE-VILLAR, J. 2008. A fully web-illustrated morphological phylogenetic study of relationships among oak gall wasps and their closest relatives (Hymenoptera: Cynipidae).

Loconte1991 LOCONTE, H. and STEVENSON, D. W. 1991. Cladistics of the Magnoliidae. Cladistics, 7, 267–296.

Longrich2010 LONGRICH, N. R., SANKEY, J. and TANKE, D. 2010. *Texacephale langstoni*, a new genus of pachycephalosaurid (Dinosauria: Ornithischia) from the upper Campanian Aguja Formation, southern Texas, USA. Cretaceous Research, 31, 274–284.

OMeara2014 O'MEARA, R. N. and THOMPSON, R. S. 2014. Were There Miocene Meridiolestidans? Assessing the phylogenetic placement of *Necrolestes patagonensis* and the presence of a 40 million year Meridiolestidan ghost lineage. Journal of Mammalian Evolution, 21, 271–284.

Rougier2012 ROUGIER, G. W., WIBLE, J. R., BECK, R. M. D. and APESTEGUIA, S. 2012. The Miocene mammal *Necrolestes* demonstrates the survival of a Mesozoic nontherian lineage into the late Cenozoic of South America. Proceedings of the National Academy of Sciences, 109, 20053–8.

Sharkey2011 SHARKEY, M. J., CARPENTER, J. M., VILHELMSEN, L., HERATY, J., LILJEBLAD, J., DOWLING, A. P. G., SCHULMEISTER, S., MURRAY, D., DEANS, A. R., RONQUIST, F., KROGMANN, L. and WHEELER, W. C. 2012. Phylogenetic relationships among superfamilies of Hymenoptera. Cladistics, 28, 80–112.

Sundue2010 SUNDUE, M. A., ISLAM, M. B. and RANKER, T. A. 2010. Systematics of Grammitid Ferns (Polypodiaceae): Using Morphology and Plastid Sequence Data to Resolve the Circumscriptions of Melpomene and the Polyphyletic Genera *Lellingeria* and *Terpsichore*. Systematic Botany, 35, 701–715.

Vinther2008 VINTHER, J., VAN ROY, P. and BRIGGS, D. E. G. 2008. Machaeridians are Palaeozoic armoured annelids. Nature, 451, 185–188.

Wilson2003 WILSON, G. D. F. and EDGECOMBE, G. D. 2003. The Triassic isopod *Protamphisopus wianamattensis* (Chilton) and comparison by extant taxa (Crustacea, Phreatoicidea). Journal of Paleontology, 77, 454–470.

Wortley2006 WORTLEY, A. H. and SCOTLAND, R. W. 2006. The effect of combining molecular and morphological data in published phylogenetic analyses. Systematic Biology, 55, 677–685.

Zanol2014 ZANOL, J., HALANYCH, K. M. and FAUCHALD, K. 2014. Reconciling taxonomy and phylogeny in the bristleworm family Eunicidae (Polychaete, Annelida). Zoologica Scripta, 43, 79–100.

Zhu2013 ZHU, M., YU, X., AHLBERG, P. E., CHOO, B., LU, J., QIAO, T., QU, Q., ZHAO, W., JIA, L., BLOM, H. and ZHU, Y. 2013. A Silurian placoderm with osteichthyan-like marginal jaw bones. Nature, 502, 188–193.

References

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

Examples

```
data("inapplicable.datasets", package = "TreeSearch")
names(inapplicable.datasets)
```

is.morphyPtr *Is an object a valid Morphy object?*

Description

Is an object a valid Morphy object?

Usage

```
is.morphyPtr(morphyObj)
```

Arguments

morphyObj	Object of class morphy, perhaps created with PhyDat2Morphy().

Value

is.morphyPtr() returns TRUE if morphyObj is a valid morphy pointer, FALSE otherwise.

Author(s)

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

See Also

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), MorphyWeights(), PhyDat2Morphy(), SingleCharMorphy(), UnloadMorphy(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

IWScore *Calculate the parsimony score of a tree given a dataset*

Description

TreeLength() uses the Morphy library (Brazeau et al. 2017) to calculate a parsimony score for a tree, handling inapplicable data according to the algorithm of Brazeau et al. (2019). Trees may be scored using equal weights, implied weights (Goloboff 1993), or profile parsimony (Faith and Trueman 2001).

Usage

```
IWScore(tree, dataset, concavity = 10L, ...)

TreeLength(tree, dataset, concavity = Inf)

## S3 method for class 'phylo'
TreeLength(tree, dataset, concavity = Inf)

## S3 method for class 'numeric'
TreeLength(tree, dataset, concavity = Inf)

## S3 method for class 'list'
TreeLength(tree, dataset, concavity = Inf)

## S3 method for class 'multiPhylo'
TreeLength(tree, dataset, concavity = Inf)

Fitch(tree, dataset)
```

Arguments

`tree`	A tree of class `phylo`, a list thereof (optionally of class `multiPhylo`), or an integer – in which case `tree` random trees will be uniformly sampled.
`dataset`	A phylogenetic data matrix of **phangorn** class `phyDat`, whose names correspond to the labels of any accompanying tree. Perhaps load into R using `ReadAsPhyDat()`. Additive (ordered) characters can be handled using `Decompose()`.
`concavity`	Determines the degree to which extra steps beyond the first are penalized. Specify a numeric value to use implied weighting (Goloboff 1993); `concavity` specifies k in $k / e + k$. A value of 10 is recommended; TNT sets a default of 3, but this is too low in some circumstances (Goloboff et al. 2018; Smith 2019). Better still explore the sensitivity of results under a range of concavity values, e.g. `k = 2 ^ (1:7)`. Specify `Inf` to weight each additional step equally, (which underperforms step weighting approaches (Goloboff et al. 2008; Goloboff et al. 2018; Goloboff and Arias 2019; Smith 2019)). Specify `"profile"` to employ an approximation of profile parsimony (Faith and Trueman 2001).
`...`	unused; allows additional parameters specified within `...` to be received by the function without throwing an error.

Value

`TreeLength()` returns a numeric vector containing the score for each tree in `tree`.

Author(s)

Martin R. Smith (using Morphy C library, by Martin Brazeau)

References

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

Brazeau MD, Smith MR, Guillerme T (2017). "MorphyLib: a library for phylogenetic analysis of categorical trait data with inapplicability." doi:10.5281/zenodo.815372.

Faith DP, Trueman JWH (2001). "Towards an inclusive philosophy for phylogenetic inference." *Systematic Biology*, **50**(3), 331–350. doi:10.1080/10635150118627.

Goloboff PA (1993). "Estimating character weights during tree search." *Cladistics*, **9**(1), 83–91. doi:10.1111/j.10960031.1993.tb00209.x.

Goloboff PA, Arias JS (2019). "Likelihood approximations of implied weights parsimony can be selected over the Mk model by the Akaike information criterion." *Cladistics*, **35**(6), 695–716. doi:10.1111/cla.12380.

Goloboff PA, Carpenter JM, Arias JS, Esquivel DRM (2008). "Weighting against homoplasy improves phylogenetic analysis of morphological data sets." *Cladistics*, **24**(5), 758–773. doi:10.1111/j.10960031.2008.00209.x.

Goloboff PA, Torres A, Arias JS (2018). "Weighted parsimony outperforms other methods of phylogenetic inference under models appropriate for morphology." *Cladistics*, **34**(4), 407–437. doi:10.1111/cla.12205.

Smith MR (2019). "Bayesian and parsimony approaches reconstruct informative trees from simulated morphological datasets." *Biology Letters*, **15**(2), 20180632. doi:10.1098/rsbl.2018.0632.

See Also

- Conduct tree search using `MaximizeParsimony()` (command line), `EasyTrees()` (graphical user interface), or `TreeSearch()` (custom optimality criteria).
- See score for each character: `CharacterLength()`.

Other tree scoring: `CharacterLength()`, `ExpectedLength()`, `LengthAdded()`, `MinimumLength()`, `MorphyTreeLength()`, `TaxonInfluence()`

Examples

```
data("inapplicable.datasets")
tree <- TreeTools::BalancedTree(inapplicable.phyData[[1]])
TreeLength(tree, inapplicable.phyData[[1]])
TreeLength(tree, inapplicable.phyData[[1]], concavity = 10)
TreeLength(tree, inapplicable.phyData[[1]], concavity = "profile")
TreeLength(5, inapplicable.phyData[[1]])
```

`Jackknife` *Jackknife resampling*

Description

Resample trees using Jackknife resampling, i.e. removing a subset of characters.

Usage

```
Jackknife(
  tree,
  dataset,
  resampleFreq = 2/3,
  InitializeData = PhyDat2Morphy,
  CleanUpData = UnloadMorphy,
  TreeScorer = MorphyLength,
  EdgeSwapper = TBRSwap,
  jackIter = 5000L,
  searchIter = 4000L,
  searchHits = 42L,
  verbosity = 1L,
  ...
)
```

Arguments

`tree`	A fully-resolved starting tree in `phylo` format, with the desired outgroup. Edge lengths are not supported and will be removed.
`dataset`	a dataset in the format required by `TreeScorer()`.
`resampleFreq`	Double between 0 and 1 stating proportion of characters to resample.
`InitializeData`	Function that sets up data object to prepare for tree search. The function will be passed the `dataset` parameter. Its return value will be passed to `TreeScorer()` and `CleanUpData()`.
`CleanUpData`	Function to destroy data object on function exit. The function will be passed the value returned by `InitializeData()`.
`TreeScorer`	function to score a given tree. The function will be passed three parameters, corresponding to the `parent` and `child` entries of a tree's edge list, and a dataset.
`EdgeSwapper`	a function that rearranges a parent and child vector, and returns a list with modified vectors; for example `SPRSwap()`.
`jackIter`	Integer specifying number of jackknife iterations to conduct.
`searchIter`	Integer specifying maximum rearrangements to perform on each bootstrap or ratchet iteration. To override this value for a single swapper function, set e.g. `attr(SwapperFunction, "searchIter") <- 99`

searchHits	Integer specifying maximum times to hit best score before terminating a tree search within a ratchet iteration. To override this value for a single swapper function, set e.g. `attr(SwapperFunction, "searchHits") <- 99`
verbosity	Numeric specifying level of detail to display in console: larger numbers provide more verbose feedback to the user.
...	further parameters to send to `TreeScorer()`

Details

The function assumes that `InitializeData()` will return a morphy object; if this doesn't hold for you, post a GitHub issue or e-mail the maintainer.

Value

`Jackknife()` returns a list of trees recovered after jackknife iterations.

Author(s)

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

See Also

- `Resample()`: Jackknife resampling for non-custom searches performed using `MaximizeParsimony()`.
- `JackLabels()`: Label nodes of a tree with jackknife supports.

Other split support functions: `ConcordanceTable()`, `JackLabels()`, `MaximizeParsimony()`, `MostContradictedFreq()`, `PresCont()`, `SiteConcordance`

Other custom search functions: `EdgeListSearch()`, `MorphyBootstrap()`, `SuccessiveApproximations()`

JackLabels *Label nodes with jackknife support values*

Description

`JackLabels()` produces a list of node labels denoting split support from a set of resampled trees, optionally printing them on a tree.

Usage

```
JackLabels(
  tree,
  jackTrees,
  plot = TRUE,
  add = FALSE,
  adj = 0,
  col = NULL,
  frame = "none",
```

```
  pos = 2L,
  showFraction = FALSE,
  format = "character",
  ...
)
```

Arguments

`tree`	A tree of class `phylo`.
`jackTrees`	A list or `multiPhylo` object containing trees generated by `Resample()` or `Jackknife()`.
`plot`	Logical specifying whether to plot results; if `FALSE`, returns blank labels for nodes near the root that do not correspond to a unique split.
`add`	Logical specifying whether to add the labels to an existing plot.
`adj`, `col`, `frame`, `pos`, `...`	Parameters to pass to `nodelabels()`.
`showFraction`	Logical specifying whether to also annotate nodes with the fraction of replicates that were decisive for the split.
`format`	Character specifying return format. `"character"` returns a character string suitable to add to the `node.labels` attribute of a tree; "numeric" returns numeric values suitable for further analysis.

Details

If an element of `jackTrees` contains multiple trees, then the iteration is counted as supporting a split if all trees contain the split, and as contradicting the split if no trees contain it. If a split is only present in a subset of trees, that iteration is considered not to be decisive, and is ignored when calculating the support for that split.

Value

A named vector specifying the proportion of jackknife iterations consistent with each node in `tree`, as plotted. If `format = "character"`, blank entries are included corresponding to nodes that do not require labels, such that the return value is in the format required by `phylo$node.label`. If multiple trees are specified per iteration, the return value has an attribute `decisive` listing, for each entry in the return value, how many iterations were decisive for that split.

Author(s)

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

See Also

Generate trees by jackknife resampling using `Resample()` for standard parsimony searches, or `Jackknife()` for custom search criteria.

Other split support functions: `ConcordanceTable()`, `Jackknife()`, `MaximizeParsimony()`, `MostContradictedFreq()`, `PresCont()`, `SiteConcordance`

Examples

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

# jackTrees will usually be generated with Jackknife() or Resample(),
# but for simplicity:
jackTrees <- as.phylo(1:100, 8)

tree <- as.phylo(0, 8)
JackLabels(tree, jackTrees)

tree$node.label <- JackLabels(tree, jackTrees, plot = FALSE)

# Write the labelled tree to screen
ape::write.tree(tree)

# Write labelled trees to a nexus file:
# write.nexus(tree, file = filename)
```

LengthAdded *Contribution of character to leaf instability*

Description

Would tree lengths change if a character was coded as ambiguous for each leaf (Pol and Escapa 2009)?

Usage

```
LengthAdded(trees, char, concavity = Inf)

PolEscapa(trees, char, concavity = Inf)
```

Arguments

`trees`	List of trees of class `phylo`, or `multiPhylo` object.
`char`	`phyDat` object containing a single character.
`concavity`	Determines the degree to which extra steps beyond the first are penalized. Specify a numeric value to use implied weighting (Goloboff 1993); `concavity` specifies k in $k / e + k$. A value of 10 is recommended; TNT sets a default of 3, but this is too low in some circumstances (Goloboff et al. 2018; Smith 2019). Better still explore the sensitivity of results under a range of concavity values, e.g. `k = 2 ^ (1:7)`. Specify `Inf` to weight each additional step equally, (which underperforms step weighting approaches (Goloboff et al. 2008; Goloboff et al. 2018; Goloboff and Arias 2019; Smith 2019)). Specify `"profile"` to employ an approximation of profile parsimony (Faith and Trueman 2001).

Details

High values for a leaf indicate that its coding contributes to instability ("wildcard" or "roguish" behaviour; see **Rogue** for further details). The coding is in tension with other data, which may indicate that the assumptions of homology that underlie the character's construction and scoring require careful scrutiny – or that the taxon in question has been subject to convergent evolution.

When inapplicable tokens are present in a character, the applicability of each coding is maintained: i.e. a leaf coded with an applicable token is never allowed to take an inapplicable value; and an inapplicable token remains inapplicable.

Value

`LengthAdded()` returns a named numeric vector listing the mean absolute change to tree length resulting if the character were coded ambiguous for each leaf in turn, under the specified concavity constant.

Author(s)

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

References

Faith DP, Trueman JWH (2001). "Towards an inclusive philosophy for phylogenetic inference." *Systematic Biology*, **50**(3), 331–350. doi:10.1080/10635150118627.

Goloboff PA (1993). "Estimating character weights during tree search." *Cladistics*, **9**(1), 83–91. doi:10.1111/j.10960031.1993.tb00209.x.

Goloboff PA, Arias JS (2019). "Likelihood approximations of implied weights parsimony can be selected over the Mk model by the Akaike information criterion." *Cladistics*, **35**(6), 695–716. doi:10.1111/cla.12380.

Goloboff PA, Carpenter JM, Arias JS, Esquivel DRM (2008). "Weighting against homoplasy improves phylogenetic analysis of morphological data sets." *Cladistics*, **24**(5), 758–773. doi:10.1111/j.10960031.2008.00209.x.

Goloboff PA, Torres A, Arias JS (2018). "Weighted parsimony outperforms other methods of phylogenetic inference under models appropriate for morphology." *Cladistics*, **34**(4), 407–437. doi:10.1111/cla.12205.

Pol D, Escapa IH (2009). "Unstable taxa in cladistic analysis: identification and the assessment of relevant characters." *Cladistics*, **25**(5), 515–527. doi:10.1111/j.10960031.2009.00258.x.

Smith MR (2019). "Bayesian and parsimony approaches reconstruct informative trees from simulated morphological datasets." *Biology Letters*, **15**(2), 20180632. doi:10.1098/rsbl.2018.0632.

See Also

Other tree scoring: `CharacterLength()`, `ExpectedLength()`, `IWScore()`, `MinimumLength()`, `MorphyTreeLength()`, `TaxonInfluence()`

Examples

```
trees <- inapplicable.trees[["Vinther2008"]]
dataset <- inapplicable.phyData[["Vinther2008"]]
char <- dataset[, 11]
added <- LengthAdded(trees, char)

PlotCharacter(
  tree = trees[[1]],
  dataset = char,
  tip.color = 1 + added[trees[[1]]$tip.label] # Colour by added steps
) -> XX # Suppress return value; display plot only
```

MaximizeParsimony *Find most parsimonious trees*

Description

Search for most parsimonious trees using the parsimony ratchet and TBR rearrangements, treating inapplicable data as such using the algorithm of Brazeau et al. (2019).

Tree search will be conducted from a specified or automatically-generated starting tree in order to find a tree with an optimal parsimony score, under implied or equal weights, treating inapplicable characters as such in order to avoid the artefacts of the standard Fitch algorithm (see Maddison 1993; Brazeau et al. 2019). Tree length is calculated using the MorphyLib C library (Brazeau et al. 2017).

Usage

```
MaximizeParsimony(
  dataset,
  tree,
  ratchIter = 7L,
  tbrIter = 2L,
  startIter = 2L,
  finalIter = 1L,
  maxHits = NTip(dataset) * 1.8,
  maxTime = 60,
  quickHits = 1/3,
  concavity = Inf,
  ratchEW = TRUE,
  tolerance = sqrt(.Machine[["double.eps"]]),
  constraint,
  verbosity = 3L
)

Resample(
  dataset,
```

```
  tree,
  method = "jack",
  proportion = 2/3,
  ratchIter = 1L,
  tbrIter = 8L,
  finalIter = 3L,
  maxHits = 12L,
  concavity = Inf,
  tolerance = sqrt(.Machine[["double.eps"]]),
  constraint,
  verbosity = 2L,
  ...
)

EasyTrees()

EasyTreesy()
```

Arguments

dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
tree	(optional) A bifurcating tree of class phylo, containing only the tips listed in dataset, from which the search should begin. If unspecified, an addition tree will be generated from dataset, respecting any supplied constraint. Edge lengths are not supported and will be deleted.
ratchIter	Numeric specifying number of iterations of the parsimony ratchet (Nixon 1999) to conduct.
tbrIter	Numeric specifying the maximum number of TBR break points on a given tree to evaluate before terminating the search. One "iteration" comprises selecting a branch to break, and evaluating each possible reconnection point in turn until a new tree improves the score. If a better score is found, then the counter is reset to zero, and tree search continues from the improved tree.
startIter	Numeric: an initial round of tree search with startIter × tbrIter TBR break points is conducted in order to locate a local optimum before beginning ratchet searches.
finalIter	Numeric: a final round of tree search will evaluate finalIter × tbrIter TBR break points, in order to sample the final optimal neighbourhood more intensely.
maxHits	Numeric specifying the maximum times that an optimal parsimony score may be hit before concluding a ratchet iteration or final search concluded.
maxTime	Numeric: after maxTime minutes, stop tree search at the next opportunity.
quickHits	Numeric: iterations on subsampled datasets will retain quickHits × maxHits trees with the best score.
concavity	Determines the degree to which extra steps beyond the first are penalized. Specify a numeric value to use implied weighting (Goloboff 1993); concavity specifies k in $k / e + k$. A value of 10 is recommended; TNT sets a default of 3,

	but this is too low in some circumstances (Goloboff et al. 2018; Smith 2019). Better still explore the sensitivity of results under a range of concavity values, e.g. `k = 2 ^ (1:7)`. Specify `Inf` to weight each additional step equally, (which underperforms step weighting approaches (Goloboff et al. 2008; Goloboff et al. 2018; Goloboff and Arias 2019; Smith 2019)). Specify `"profile"` to employ an approximation of profile parsimony (Faith and Trueman 2001).
`ratchEW`	Logical specifying whether to use equal weighting during ratchet iterations, improving search speed whilst still facilitating escape from local optima.
`tolerance`	Numeric specifying degree of suboptimality to tolerate before rejecting a tree. The default, `sqrt(.Machine$double.eps)`, retains trees that may be equally parsimonious but for rounding errors. Setting to larger values will include trees suboptimal by up to `tolerance` in search results, which may improve the accuracy of the consensus tree (at the expense of resolution) (Smith 2019).
`constraint`	Either an object of class `phyDat`, in which case returned trees will be perfectly compatible with each character in `constraint`; or a tree of class `phylo`, all of whose nodes will occur in any output tree. See `ImposeConstraint()` and vignette for further examples.
`verbosity`	Integer specifying level of messaging; higher values give more detailed commentary on search progress. Set to `0` to run silently.
`method`	Unambiguous abbreviation of `jackknife` or `bootstrap` specifying how to resample characters. Note that jackknife is considered to give more meaningful results.
`proportion`	Numeric between 0 and 1 specifying what proportion of characters to retain under jackknife resampling.
`...`	Additional parameters to `MaximizeParsimony()`.

Details

Tree search commences with `ratchIter` iterations of the parsimony ratchet (Nixon 1999), which bootstraps the input dataset in order to escape local optima. A final round of tree bisection and reconnection (`TBR`) is conducted to broaden the sampling of trees.

This function can be called using the R command line / terminal, or through the "shiny" graphical user interface app (type `EasyTrees()` to launch).

The optimal strategy for tree search depends in part on how close to optimal the starting tree is, the size of the search space (which increases super-exponentially with the number of leaves), and the complexity of the search space (e.g. the existence of multiple local optima).

One possible approach is to employ four phases:

1. Rapid search for local optimum: tree score is typically easy to improve early in a search, because the initial tree is often far from optimal. When many moves are likely to be accepted, running several rounds of search with a low value of `maxHits` and a high value of `tbrIter` allows many trees to be evaluated quickly, hopefully moving quickly to a more promising region of tree space.
2. Identification of local optimum: Once close to a local optimum, a more extensive search with a higher value of `maxHits` allows a region to be explored in more detail. Setting a high value of `tbrIter` will search a local neighbourhood more completely

3. Search for nearby peaks: Ratchet iterations allow escape from local optima. Setting `ratchIter` to a high value searches the wider neighbourhood more extensively for other nearby peaks; `ratchEW = TRUE` accelerates these exploratory searches. Ratchet iterations can be ineffective when `maxHits` is too low for the search to escape its initial location.
4. Extensive search of final optimum. As with step 2, it may be valuable to fully explore the optimum that is found after ratchet searches to be sure that the locally optimal score has been obtained. Setting a high value of `finalIter` performs a thorough search that can give confidence that further searches would not find better (local) trees.

A search is unlikely to have found a global optimum if:

- Tree score continues to improve on the final iteration. If a local optimum has not yet been reached, it is unlikely that a global optimum has been reached. Try increasing `maxHits`.
- Successive ratchet iterations continue to improve tree scores. If a recent ratchet iteration improved the score, rather than finding a different region of tree space with the same optimal score, it is likely that still better global optima remain to be found. Try increasing `ratchIter` (more iterations give more chance for improvement) and `maxHits` (to get closer to the local optimum after each ratchet iteration).
- Optimal areas of tree space are only visited by a single ratchet iteration. (See vignette: Exploring tree space.) If some areas of tree space are only found by one ratchet iteration, there may well be other, better areas that have not yet been visited. Try increasing `ratchIter`.

When continuing a tree search, it is usually best to start from an optimal tree found during the previous iteration - there is no need to start from scratch.

A more time consuming way of checking that a global optimum has been reached is to repeat a search with the same parameters multiple times, starting from a different, entirely random tree each time. If all searches obtain the same optimal tree score despite their different starting points, this score is likely to correspond to the global optimum.

For detailed documentation of the "TreeSearch" package, including full instructions for loading phylogenetic data into R and initiating and configuring tree search, see the package documentation.

Value

`MaximizeParsimony()` returns a list of trees with class `multiPhylo`. This lists all trees found during each search step that are within `tolerance` of the optimal score, listed in the sequence that they were first visited, and named according to the step in which they were first found; it may contain more than `maxHits` elements. Note that the default search parameters may need to be increased in order for these trees to be the globally optimal trees; examine the messages printed during tree search to evaluate whether the optimal score has stabilized.

The return value has the attribute `firstHit`, a named integer vector listing the number of optimal trees visited for the first time in each stage of the tree search. Stages are named:

- `seed`: starting trees;
- `start`: Initial TBR search;
- `ratchN`: Ratchet iteration `N`;
- `final`: Final TBR search. The first tree hit for the first time in ratchet iteration three is named `ratch3_1`.

`Resample()` returns a `multiPhylo` object containing a list of trees obtained by tree search using a resampled version of `dataset`.

Resampling

Note that bootstrap support is a measure of the amount of data supporting a split, rather than the amount of confidence that should be afforded the grouping. "Bootstrap support of 100% is not enough, the tree must also be correct" (Phillips et al. 2004). See discussion in Egan (2006); Wagele et al. (2009); (Simmons and Freudenstein 2011); Kumar et al. (2012).

For a discussion of suitable search parameters in resampling estimates, see Muller (2005). The user should decide whether to start each resampling from the optimal tree (which may be quicker, but result in overestimated support values as searches get stuck in local optima close to the optimal tree) or a random tree (which may take longer as more rearrangements are necessary to find an optimal tree on each iteration).

For other ways to estimate clade concordance, see `SiteConcordance()`.

Author(s)

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

References

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

Brazeau MD, Smith MR, Guillerme T (2017). "MorphyLib: a library for phylogenetic analysis of categorical trait data with inapplicability." doi:10.5281/zenodo.815372.

Egan MG (2006). "Support versus corroboration." *Journal of Biomedical Informatics*, **39**(1), 72–85. doi:10.1016/j.jbi.2005.11.007.

Faith DP, Trueman JWH (2001). "Towards an inclusive philosophy for phylogenetic inference." *Systematic Biology*, **50**(3), 331–350. doi:10.1080/10635150118627.

Goloboff PA (1993). "Estimating character weights during tree search." *Cladistics*, **9**(1), 83–91. doi:10.1111/j.10960031.1993.tb00209.x.

Goloboff PA, Arias JS (2019). "Likelihood approximations of implied weights parsimony can be selected over the Mk model by the Akaike information criterion." *Cladistics*, **35**(6), 695–716. doi:10.1111/cla.12380.

Goloboff PA, Carpenter JM, Arias JS, Esquivel DRM (2008). "Weighting against homoplasy improves phylogenetic analysis of morphological data sets." *Cladistics*, **24**(5), 758–773. doi:10.1111/j.10960031.2008.00209.x.

Goloboff PA, Torres A, Arias JS (2018). "Weighted parsimony outperforms other methods of phylogenetic inference under models appropriate for morphology." *Cladistics*, **34**(4), 407–437. doi:10.1111/cla.12205.

Kumar S, Filipski AJ, Battistuzzi FU, Kosakovsky Pond SL, Tamura K (2012). "Statistics and truth in phylogenomics." *Molecular Biology and Evolution*, **29**(2), 457–472. doi:10.1093/molbev/msr202.

Maddison WP (1993). "Missing data versus missing characters in phylogenetic analysis." *Systematic Biology*, **42**(4), 576–581. doi:10.1093/sysbio/42.4.576.

Muller KF (2005). "The efficiency of different search strategies in estimating parsimony jackknife, bootstrap, and Bremer support." *BMC Evolutionary Biology*, **5**(1), 58. doi:10.1186/14712148558.

Nixon KC (1999). "The Parsimony Ratchet, a new method for rapid parsimony analysis." *Cladistics*, **15**(4), 407–414. ISSN 0748-3007. doi:10.1111/j.10960031.1999.tb00277.x.

Phillips MJ, Delsuc F, Penny D (2004). "Genome-scale phylogeny and the detection of systematic biases." *Molecular biology and evolution*, **21**(7), 1455–8. doi:10.1093/molbev/msh137.

Simmons MP, Freudenstein JV (2011). "Spurious 99% bootstrap and jackknife support for unsupported clades." *Molecular Phylogenetics and Evolution*, **61**(1), 177–191. doi:10.1016/j.ympev.2011.06.003.

Smith MR (2019). "Bayesian and parsimony approaches reconstruct informative trees from simulated morphological datasets." *Biology Letters*, **15**(2), 20180632. doi:10.1098/rsbl.2018.0632.

Wagele JW, Letsch H, Klussmann-Kolb A, Mayer C, Misof B, Wagele H (2009). "Phylogenetic support values are not necessarily informative: the case of the Serialia hypothesis (a mollusk phylogeny)." *Frontiers in Zoology*, **6**(1), 12–29. doi:10.1186/17429994612.

See Also

Tree search *via* graphical user interface: `EasyTrees()`

Other split support functions: `ConcordanceTable()`, `JackLabels()`, `Jackknife()`, `MostContradictedFreq()`, `PresCont()`, `SiteConcordance`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {
  # launch "shiny" point-and-click interface
  EasyTrees()

  # Here too, use the "continue search" function to ensure that tree score
  # has stabilized and a global optimum has been found
}


# Load data for analysis in R
library("TreeTools")
data("inapplicable.phyData", package = "TreeSearch")
dataset <- inapplicable.phyData[["Asher2005"]]
```

```
# A very quick run for demonstration purposes
trees <- MaximizeParsimony(dataset, ratchIter = 0, startIter = 0,
                           tbrIter = 1, maxHits = 4, maxTime = 1/100,
                           concavity = 10, verbosity = 4)
names(trees)
cons <- Consensus(trees)

# In actual use, be sure to check that the score has converged on a global
# optimum, conducting additional iterations and runs as necessary.

if (interactive()) {
# Jackknife resampling
nReplicates <- 10
jackTrees <- replicate(nReplicates,
  #c() ensures that each replicate returns a list of trees
  c(Resample(dataset, trees, ratchIter = 0, tbrIter = 2, startIter = 1,
             maxHits = 5, maxTime = 1 / 10,
             concavity = 10, verbosity = 0))
 )

# In a serious analysis, more replicates would be conducted, and each
# search would undergo more iterations.

# Now we must decide what to do with the multiple optimal trees from
# each replicate.

# Set graphical parameters for plotting
oPar <- par(mar = rep(0, 4), cex = 0.9)

# Take the strict consensus of all trees for each replicate
# (May underestimate support)
JackLabels(cons, lapply(jackTrees, ape::consensus))

# Take a single tree from each replicate (here, the first)
# Potentially problematic if chosen tree is not representative
JackLabels(cons, lapply(jackTrees, `[[`, 1))

# Count iteration as support if all most parsimonious trees support a split;
# as contradiction if all trees contradict it; don't include replicates where
# not all trees agree on the resolution of a split.
labels <- JackLabels(cons, jackTrees)

# How many iterations were decisive for each node?
attr(labels, "decisive")

# Show as proportion of decisive iterations
JackLabels(cons, jackTrees, showFrac = TRUE)

# Restore graphical parameters
par(oPar)
}

# Tree search with a constraint
```

```
constraint <- MatrixToPhyDat(c(a = 1, b = 1, c = 0, d = 0, e = 0, f = 0))
characters <- MatrixToPhyDat(matrix(
  c(0, 1, 1, 1, 0, 0,
    1, 1, 1, 0, 0, 0), ncol = 2,
  dimnames = list(letters[1:6], NULL)))
MaximizeParsimony(characters, constraint = constraint, verbosity = 0)
```

MinimumLength *Minimum and Maximum lengths possible for a character*

Description

The smallest and largest length that a phylogenetic character can attain on any tree.

Usage

```
MinimumLength(x, compress = FALSE)

## S3 method for class 'phyDat'
MinimumLength(x, compress = FALSE)

## S3 method for class 'numeric'
MinimumLength(x, compress = NA)

## S3 method for class 'character'
MinimumLength(x, compress = TRUE)

## S3 method for class 'character'
MaximumLength(x, compress = TRUE)

MinimumSteps(x)

MaximumLength(x, compress = TRUE)

## S3 method for class 'numeric'
MaximumLength(x, compress = NA)
```

Arguments

x	An object of class phyDat; or a string to be coerced to a phyDat object via TreeTools::StringToPhyDat(); or an integer vector listing the tokens that may be present at each tip along a single character, with each token represented as a binary digit; e.g. a value of 11 (= 2^0 + 2^1 + 2^3) means that the tip may have tokens 0, 1 or 3. Inapplicable tokens should be denoted with the integer 0 (not 2^0).
compress	Logical specifying whether to retain the compression of a phyDat object or to return a vector specifying to each individual character, decompressed using the dataset's index attribute.

Details

Ambiguous inapplicable states (e.g. {0, -}) are currently replaced with the plain inapplicable token -, reflecting the current behaviour of Morphy.

Value

MinimumLength() returns a vector of integers specifying the minimum number of steps that each character must contain.

MaximumLength() returns a vector of integers specifying the maximum number of steps that each character can attain in a parsimonious reconstruction on a tree. Inapplicable tokens are not yet supported.

Author(s)

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

See Also

Other tree scoring: CharacterLength(), ExpectedLength(), IWScore(), LengthAdded(), MorphyTreeLength(), TaxonInfluence()

Examples

```
data("inapplicable.datasets")
myPhyDat <- inapplicable.phyData[[4]]

# load your own data with
# my.PhyDat <- as.phyDat(read.nexus.data("filepath"))
# or Windows users can select a file interactively using:
# my.PhyDat <- as.phyDat(read.nexus.data(choose.files()))

class(myPhyDat) # phyDat object

# Minimum length of each character in turn
MinimumLength(myPhyDat)

# Collapse duplicate characters, per phyDat compression
MinimumLength(myPhyDat, compress = TRUE)

# Calculate length of a single character from its textual representation
MinimumLength("-{-1}{-2}{-3}2233")
MaximumLength("----0011")
```

MorphyBootstrap *Parsimony Ratchet*

Description

`Ratchet()` uses the parsimony ratchet (Nixon 1999) to search for a more parsimonious tree using custom optimality criteria.

Usage

```
MorphyBootstrap(
  edgeList,
  morphyObj,
  EdgeSwapper = NNISwap,
  maxIter,
  maxHits,
  verbosity = 1L,
  stopAtPeak = FALSE,
  stopAtPlateau = 0L,
  ...
)

Ratchet(
  tree,
  dataset,
  InitializeData = PhyDat2Morphy,
  CleanUpData = UnloadMorphy,
  TreeScorer = MorphyLength,
  Bootstrapper = MorphyBootstrap,
  swappers = list(TBRSwap, SPRSwap, NNISwap),
 BootstrapSwapper = if (is.list(swappers)) swappers[[length(swappers)]] else swappers,
  returnAll = FALSE,
  stopAtScore = NULL,
  stopAtPeak = FALSE,
  stopAtPlateau = 0L,
  ratchIter = 100,
  ratchHits = 10,
  searchIter = 4000,
  searchHits = 42,
  bootstrapIter = searchIter,
  bootstrapHits = searchHits,
  verbosity = 1L,
  suboptimal = sqrt(.Machine[["double.eps"]]),
  ...
)

MultiRatchet(
```

```
  tree,
  dataset,
  ratchHits = 10,
  searchIter = 500,
  searchHits = 20,
  verbosity = 0L,
  swappers = list(RootedNNISwap),
  nSearch = 10,
  stopAtScore = NULL,
  ...
)

RatchetConsensus(
  tree,
  dataset,
  ratchHits = 10,
  searchIter = 500,
  searchHits = 20,
  verbosity = 0L,
  swappers = list(RootedNNISwap),
  nSearch = 10,
  stopAtScore = NULL,
  ...
)
```

Arguments

`edgeList`	a list containing the following: • vector of integers corresponding to the parent of each edge in turn • vector of integers corresponding to the child of each edge in turn • (optionally) score of the tree • (optionally, if score provided) number of times this score has been hit
`morphyObj`	Object of class `morphy`, perhaps created with `PhyDat2Morphy()`.
`EdgeSwapper`	a function that rearranges a parent and child vector, and returns a list with modified vectors; for example `SPRSwap()`.
`maxIter`	Numeric specifying maximum number of iterations to perform in tree search.
`maxHits`	Numeric specifying maximum number of hits to accomplish in tree search.
`verbosity`	Numeric specifying level of detail to display in console: larger numbers provide more verbose feedback to the user.
`stopAtPeak`	Logical specifying whether to terminate search once a subsequent iteration recovers a sub-optimal score. Will be overridden if a passed function has an attribute `stopAtPeak` set by `attr(FunctionName, "stopAtPeak") <- TRUE`.
`stopAtPlateau`	Integer. If > 0, tree search will terminate if the score has not improved after `stopAtPlateau` iterations. Will be overridden if a passed function has an attribute `stopAtPlateau` set by `attr(FunctionName, "stopAtPlateau") <- TRUE`.

`...`	further parameters to send to `TreeScorer()`
`tree`	A fully-resolved starting tree in `phylo` format, with the desired outgroup. Edge lengths are not supported and will be removed.
`dataset`	a dataset in the format required by `TreeScorer()`.
`InitializeData`	Function that sets up data object to prepare for tree search. The function will be passed the `dataset` parameter. Its return value will be passed to `TreeScorer()` and `CleanUpData()`.
`CleanUpData`	Function to destroy data object on function exit. The function will be passed the value returned by `InitializeData()`.
`TreeScorer`	function to score a given tree. The function will be passed three parameters, corresponding to the `parent` and `child` entries of a tree's edge list, and a dataset.
`Bootstrapper`	Function to perform bootstrapped rearrangements of tree. First arguments will be an `edgeList` and a dataset, initialized using `InitializeData()`. Should return a rearranged `edgeList`.
`swappers`	A list of functions to use to conduct edge rearrangement during tree search. Provide functions like `NNISwap` to shuffle root position, or `RootedTBRSwap` if the position of the root should be retained. You may wish to use extreme swappers (such as TBR) early in the list, and a more subtle rearranger (such as NNI) later in the list to make incremental tinkerings once an almost-optimal tree has been found.
`BootstrapSwapper`	Function such as `RootedNNISwap` to use to rearrange trees within `Bootstrapper()`.
`returnAll`	Set to `TRUE` to report all MPTs encountered during the search, perhaps to analyse consensus.
`stopAtScore`	stop search as soon as this score is hit or beaten.
`ratchIter`	Stop when this many ratchet iterations have been performed.
`ratchHits`	Stop when this many ratchet iterations have found the same best score.
`searchIter`	Integer specifying maximum rearrangements to perform on each bootstrap or ratchet iteration. To override this value for a single swapper function, set e.g. `attr(SwapperFunction, "searchIter") <- 99`
`searchHits`	Integer specifying maximum times to hit best score before terminating a tree search within a ratchet iteration. To override this value for a single swapper function, set e.g. `attr(SwapperFunction, "searchHits") <- 99`
`bootstrapIter`	Integer specifying maximum rearrangements to perform on each bootstrap iteration (default: `searchIter`).
`bootstrapHits`	Integer specifying maximum times to hit best score on each bootstrap iteration (default: `searchHits`).
`suboptimal`	retain trees that are suboptimal by this score. Defaults to a small value that will counter rounding errors.
`nSearch`	Number of Ratchet searches to conduct (for `RatchetConsensus()`)

Details

For usage pointers, see the vignette.

Value

MorphyBootstrap() returns a tree that is optimal under a random sampling of the original characters.

Ratchet() returns a tree modified by parsimony ratchet iterations.

MultiRatchet() returns a list of optimal trees produced by nSearch Ratchet() searches, from which a consensus tree can be generated using ape::consensus() or TreeTools::ConsensusWithout().

Functions

- RatchetConsensus(): deprecated alias for MultiRatchet()

Author(s)

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

References

Nixon KC (1999). "The Parsimony Ratchet, a new method for rapid parsimony analysis." *Cladistics*, **15**(4), 407–414. ISSN 0748-3007. doi:10.1111/j.10960031.1999.tb00277.x.

See Also

- Adapted from pratchet() in the **phangorn** package.

Other custom search functions: EdgeListSearch(), Jackknife(), SuccessiveApproximations()

Examples

```
data("Lobo", package = "TreeTools")
njtree <- TreeTools::NJTree(Lobo.phy)
# Increase value of ratchIter and searchHits to do a proper search
quickResult <- Ratchet(njtree, Lobo.phy, ratchIter = 2, searchHits = 3)

# Plot result (legibly)
oldPar <- par(mar = rep(0, 4), cex = 0.75)
plot(quickResult)
par(oldPar)
```

MorphyWeights *Set and get the character weightings associated with a Morphy object.*

Description

MorphyWeights() details the approximate and exact weights associated with characters in a Morphy object; SetMorphyWeights() edits them.

Usage

```
MorphyWeights(morphyObj)

SetMorphyWeights(weight, morphyObj, checkInput = TRUE)
```

Arguments

morphyObj	Object of class morphy, perhaps created with PhyDat2Morphy().
weight	A vector listing the new weights to be applied to each character
checkInput	Whether to sanity-check input data before applying. Defaults to TRUE to protect the user from crashes.

Value

MorphyWeights() returns a data frame with two named rows and one column per character pattern: row 1, approx, is a list of integers specifying the approximate (integral) weights used by MorphyLib; row 2, exact, is a list of numerics specifying the exact weights specified by the user.

SetMorphyWeights() returns the Morphy error code generated when applying weight.

Author(s)

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

See Also

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), PhyDat2Morphy(), SingleCharMorphy(), UnloadMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

Examples

```
tokens <- matrix(c(
  0, 0, 0, 1, 1, 2,
  0, 0, 0, 0, 0, 0), byrow = TRUE, nrow = 2L,
  dimnames = list(letters[1:2], NULL))
pd <- TreeTools::MatrixToPhyDat(tokens)
morphyObj <- PhyDat2Morphy(pd)
MorphyWeights(morphyObj)
if (SetMorphyWeights(c(1, 1.5, 2/3), morphyObj) != 0L) message("Errored")
MorphyWeights(morphyObj)
morphyObj <- UnloadMorphy(morphyObj)
```

`MostContradictedFreq` *Frequency of most common contradictory split*

Description

`MostContradictedFreq()` counts the occurrences of the single split that most frequently contradicts each split in `tree`.

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

```
MostContradictedFreq(tree, forest)
```

Arguments

`tree`	A tree of class `phylo`.
`forest`	a list of trees of class `phylo`, or a `multiPhylo` object; or a `Splits` object.

Details

Goloboff et al. (2003) propose comparing the frequency of a split in a resampled population with the frequency of the most common contradictory split. This measure contributes to the "groups present / contradicted" score.

Value

`MostContradictedFreq()` returns, for each split in `tree`, the number of times that its most common contradictory split occurs in `forest`.

See Also

`PresCont()` calculates the "groups present / contradicted" score.

Other split support functions: `ConcordanceTable()`, `JackLabels()`, `Jackknife()`, `MaximizeParsimony()`, `PresCont()`, `SiteConcordance`

NNI *Nearest neighbour interchange (NNI)*

Description

`NNI()`performs a single iteration of the nearest-neighbour interchange algorithm; `RootedNNI()` retains the position of the root. These functions are based on equivalents in the **phangorn** package. `cNNI()` is an equivalent function coded in C, that runs much faster.

Usage

```
NNI(tree, edgeToBreak = NULL)

cNNI(tree, edgeToBreak = NULL, whichSwitch = NULL)

NNISwap(parent, child, nTips = (length(parent)/2L) + 1L, edgeToBreak = NULL)

RootedNNI(tree, edgeToBreak = NULL)

RootedNNISwap(
  parent,
  child,
  nTips = (length(parent)/2L) + 1L,
  edgeToBreak = NULL
)
```

Arguments

`tree`	A tree of class `phylo`. For `cNNI()`, this must be a binary tree rooted on a single leaf, whose root node is the lowest numbered internal node.
`edgeToBreak`	In `(Rooted/Double)NNI()`, an optional integer specifying the index of an edge to rearrange, generated randomly if not specified. If `-1`, a complete list of all trees one step from the input tree will be returned. In `cNNI()`, an integer from zero to `nEdge(tree) - nTip(tree) - 2`, specifying which internal edge to break.
`whichSwitch`	Integer from zero to one, specifying which way to re-build the broken internal edge.
`parent`	Integer vector corresponding to the first column of the edge matrix of a tree of class `phylo`, i.e. `tree[["edge"]][, 1]`
`child`	Integer vector corresponding to the second column of the edge matrix of a tree of class `phylo`, i.e. `tree[["edge"]][, 2]`.
`nTips`	(optional) Number of tips.

Details

Branch lengths are not supported.

All nodes in a tree must be bifurcating; `ape::collapse.singles()` and `ape::multi2di()` may help.

Value

Returns a tree with class phylo (if returnAll = FALSE) or a set of trees, with class multiPhylo (if returnAll = TRUE).

cNNI() returns a tree of class phylo, rooted on the same leaf, on which the specified rearrangement has been conducted.

NNISwap() returns a list containing two elements, corresponding in turn to the rearranged parent and child parameters.

a list containing two elements, corresponding in turn to the rearranged parent and child parameters

Functions

- NNISwap(): faster version that takes and returns parent and child parameters
- RootedNNI(): Perform NNI rearrangement, retaining position of root
- RootedNNISwap(): faster version that takes and returns parent and child parameters

Author(s)

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

References

The algorithm is summarized in Felsenstein J (2004). *Inferring phylogenies*. Sinauer Associates, Sunderland, Massachusetts.

See Also

Other tree rearrangement functions: SPR(), TBR()

Examples

```
tree <- TreeTools::BalancedTree(8)
# A random rearrangement
NNI(tree)
cNNI(tree)

# All trees one NNI rearrangement away
NNI(tree, edgeToBreak = -1)

# Manual random sampling
cNNI(tree, sample.int(14 - 8 - 1, 1), sample.int(2, 1))

# A specified rearrangement
cNNI(tree, 0, 0)

# If a tree may not be binary, collapse nodes with
tree <- TreeTools::MakeTreeBinary(tree)

# If a tree may be improperly rooted, use
tree <- TreeTools::RootTree(tree, 1)
```

```
# If a tree may exhibit unusual node ordering, this can be addressed with
tree <- TreeTools::Preorder(tree)
```

PhyDat2Morphy *Initialize a Morphy object from a* phyDat *object*

Description

Creates a new Morphy object with the same size and characters as the phyDat object. Once finished with the object, it should be destroyed using UnloadMorphy() to free the allocated memory.

Usage

```
PhyDat2Morphy(phy, gap = "inapplicable", weight = attr(phy, "weight"))
```

Arguments

phy	An object of **phangorn** class phyDat.
gap	An unambiguous abbreviation of inapplicable, ambiguous (= missing), or extra state, specifying how gaps will be handled.
weight	Numeric giving weight to apply to each character. Will be recycled.

Value

PhyDat2Morphy() returns a pointer to an initialized Morphy object.

Author(s)

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

See Also

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), MorphyWeights(), SingleCharMorphy(), UnloadMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

Examples

```
data("Lobo", package="TreeTools")
morphyObj <- PhyDat2Morphy(Lobo.phy)
# Set object to be destroyed at end of session or closure of function
# on.exit(morphyObj <- UnloadMorphy(morphyObj), add = TRUE)

# Do something with pointer
# ....

# Or, instead of on.exit, manually destroy morphy object and free memory:
morphyObj <- UnloadMorphy(morphyObj)
```

PlotCharacter *Plot the distribution of a character on a tree*

Description

Reconstructs the distribution of a character on a tree topology using the modified Fitch algorithm presented in Brazeau et al. (2019).

Usage

```
PlotCharacter(
  tree,
  dataset,
  char = 1L,
  updateTips = FALSE,
  plot = TRUE,
  tokenCol = NULL,
  ambigCol = "grey",
  inappCol = "lightgrey",
  ambigLty = "dotted",
  inappLty = "dashed",
  plainLty = par("lty"),
  tipOffset = 1,
  unitEdge = FALSE,
  Display = function(tree) tree,
  ...
)

## S3 method for class 'phylo'
PlotCharacter(
  tree,
  dataset,
  char = 1L,
  updateTips = FALSE,
  plot = TRUE,
```

```
  tokenCol = NULL,
  ambigCol = "grey",
  inappCol = "lightgrey",
  ambigLty = "dotted",
  inappLty = "dashed",
  plainLty = par("lty"),
  tipOffset = 1,
  unitEdge = FALSE,
  Display = function(tree) tree,
  ...
)

## S3 method for class 'multiPhylo'
PlotCharacter(
  tree,
  dataset,
  char = 1L,
  updateTips = FALSE,
  plot = TRUE,
  tokenCol = NULL,
  ambigCol = "grey",
  inappCol = "lightgrey",
  ambigLty = "dotted",
  inappLty = "dashed",
  plainLty = par("lty"),
  tipOffset = 1,
  unitEdge = FALSE,
  Display = function(tree) tree,
  ...
)

## S3 method for class 'list'
PlotCharacter(
  tree,
  dataset,
  char = 1L,
  updateTips = FALSE,
  plot = TRUE,
  tokenCol = NULL,
  ambigCol = "grey",
  inappCol = "lightgrey",
  ambigLty = "dotted",
  inappLty = "dashed",
  plainLty = par("lty"),
  tipOffset = 1,
  unitEdge = FALSE,
  Display = function(tree) tree,
  ...
```

)

Arguments

tree	A bifurcating tree of class phylo, or a list or multiPhylo object containing such trees.
dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
char	Index of character to plot.
updateTips	Logical; if FALSE, tips will be labelled with their original state in dataset.
plot	Logical specifying whether to plot the output.
tokenCol	Palette specifying colours to associate with each token in turn, in the sequence listed in attr(dataset, "levels").
ambigCol, ambigLty, inappCol, inappLty, plainLty	Colours and line types to apply to ambiguous, inapplicable and applicable tokens. See the lty graphical parameter for details of line styles. Overrides tokenCol.
tipOffset	Numeric: how much to offset tips from their labels.
unitEdge	Logical: Should all edges be plotted with a unit length?
Display	Function that takes argument tree and returns a tree of class phylo, formatted as it will be plotted.
...	Further arguments to pass to plot.phylo().

Value

PlotCharacter() invisibly returns a matrix in which each row corresponds to a numbered tip or node of tree, and each column corresponds to a token; the tokens that might parsimoniously be present at each point on a tree are denoted with TRUE. If multiple trees are supplied, the strict consensus of all trees and reconstructions will be returned; i.e. if a node is reconstructed as 0 in one tree, and 2 in another, it will be labelled (02).

Author(s)

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

References

Brazeau MD, Guillerme T, Smith MR (2019). "An algorithm for morphological phylogenetic analysis with inapplicable data." *Systematic Biology*, **68**(4), 619–631. doi:10.1093/sysbio/syy083.

Examples

```
# Set up plotting area
oPar <- par(mar = rep(0, 4))

tree <- ape::read.tree(text =
```

```
  "((((((a, b), c), d), e), f), (g, (h, (i, (j, (k, l))))));")
## A character with inapplicable data
dataset <- TreeTools::StringToPhyDat("23--1??--032", tips = tree)
plotted <- PlotCharacter(tree, dataset)
plotted

# Character from a real dataset
data("Lobo", package = "TreeTools")
dataset <- Lobo.phy
tree <- TreeTools::NJTree(dataset)
PlotCharacter(tree, dataset, 14)
par(oPar)
```

`PrepareDataProfile` *Prepare data for Profile Parsimony*

Description

Calculates profiles for each character in a dataset. Will also simplify characters, with a warning, where they are too complex for the present implementation of profile parsimony:

- inapplicable tokens will be replaced with the ambiguous token (i.e. `-` → ?);
- Ambiguous tokens will be treated as fully ambiguous (i.e. `{02}` → ?)
- Where more than two states are informative (i.e. unambiguously present in more than one taxon), states beyond the two most informative will be ignored.

Usage

```
PrepareDataProfile(dataset)

PrepareDataIW(dataset)
```

Arguments

`dataset`	dataset of class `phyDat`

Value

An object of class `phyDat`, with additional attributes. `PrepareDataProfile` adds the attributes:

- `info.amounts`: details the information represented by each character when subject to N additional steps.
- `informative`: logical specifying which characters contain any phylogenetic information.
- `bootstrap`: The character vector `c("info.amounts", "split.sizes")`, indicating attributes to sample when bootstrapping the dataset (e.g. in Ratchet searches).

`PrepareDataIW` adds the attribute:

- `min.length`: The minimum number of steps that must be present in each transformation series.

Functions

- `PrepareDataIW()`: Prepare data for implied weighting

Author(s)

Martin R. Smith; written with reference to `phangorn:::prepareDataFitch()`

See Also

Other profile parsimony functions: `Carter1()`, `StepInformation()`, `WithOneExtraStep()`, `profiles`

Examples

```
data("congreveLamsdellMatrices")
dataset <- congreveLamsdellMatrices[[42]]
PrepareDataProfile(dataset)
```

PresCont *Group present or contradicted score*

Description

Implements the Groups Present / Contradicted (GC) measure (Goloboff et al. 2003).

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

```
PresCont(
  tree,
  forest,
  plot = TRUE,
  add = FALSE,
  adj = 0,
  col = NULL,
  frame = "none",
  pos = 2L,
  ...
)
```

Arguments

`tree`	A tree of class `phylo`.
`forest`	a list of trees of class `phylo`, or a `multiPhylo` object; or a `Splits` object.
`plot`	Logical specifying whether to plot results; if `FALSE`, returns blank labels for nodes near the root that do not correspond to a unique split.

add Logical specifying whether to add the labels to an existing plot.

adj, col, frame, pos, ... Parameters to pass to nodelabels().

Details

The GC score ranges from -1 to 1, and is intended as an alternative to bootstrap or jackknife support values.

The GC score counts the number of trees in forest that include a given split, and subtracts the number of times that the *most frequent* contradictory split occurs. This value is then divided by the number of trees in forest.

A score of 1 denotes that every tree in a forest (typically of bootstrap or jackknife replicates) contains the split in question. A score of -1 denotes that a specific contradictory split occurs in every tree in forest. A score of zero indicates no support: i.e. that the split exhibits no more support than its most common contradictory split.

The most frequent contradictory split is used to discriminate between a scenario where a given split enjoys much more support than any other alternative (even if many alternatives exist, each with low support), and a scenario where the chosen split is scarcely any better supported than a competing alternative. The split is considered better supported than the latter, where the runner-up may become preferred with a modest change to the underlying dataset.

Value

PresCont() returns a character vector that labels the nodes in tree in sequence, suitable for passing to nodelabels() or tree$node.label. If plot = TRUE, it also plots tree, with splits labelled by their groups present / contradicted score.

Author(s)

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

References

Goloboff PA, Farris JS, Källersjö M, Oxelman B, Ramírez MJ, Szumik CA (2003). "Improvements to Resampling Measures of Group Support." *Cladistics*, **19**(4), 324–332. doi:10.1016/S0748-3007(03)000604.

See Also

SplitFrequency() and MostContradictedFreq() will compute the number of trees that contain the split, and the frequency of the most contradicted split, respectively.

Other split support functions: ConcordanceTable(), JackLabels(), Jackknife(), MaximizeParsimony(), MostContradictedFreq(), SiteConcordance

Examples

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

# jackTrees will usually be generated with Jackknife() or Resample(),
# but for simplicity:
jackTrees <- as.phylo(1:100, 8)

tree <- as.phylo(0, 8)
PresCont(tree, jackTrees)

tree$node.label <- PresCont(tree, jackTrees, plot = FALSE)

# Write the labelled tree to screen
ape::write.tree(tree)

# Write labelled trees to a nexus file:
# write.nexus(tree, file = filename)

# Compute the measure for further analysis
gpc <- (SplitFrequency(tree, jackTrees) -
  MostContradictedFreq(tree, jackTrees)) / length(jackTrees)
gpc
```

---

profiles *Empirically counted profiles for small trees*

Description

The base 2 logarithm of the number of trees containing *s* steps, calculated by scoring a character on each *n*-leaf tree.

Usage

```
profiles
```

Format

A list with the structure `profiles[[number of leaves]][[number of tokens]][[tokens in smallest split]]` The list entry returns a named numeric vector; each entry lists log2(proportion of *n*-leaf trees with *s* or fewer steps for this character).

See Also

Other profile parsimony functions: `Carter1()`, `PrepareDataProfile()`, `StepInformation()`, `WithOneExtraStep()`

Examples

```
data(profiles)

# Load profile for a character of the structure 0 0 0 1 1 1 1 1
profile3.5 <- profiles[[8]][[2]][[3]]

# Number of trees with _s_ or fewer steps on that character
TreeTools::NUnrooted(8) * 2 ^ profile3.5
```

QACol *Generate colour to depict the amount and quality of observations*

Description

Generate colour to depict the amount and quality of observations

Usage

```
QACol(quality, amount)

QCol(quality, amount)

QALegend(
  where = c(0.1, 0.3, 0.1, 0.3),
  n = 5,
  Col = QACol,
  xlab = "Amount →",
  ylab = "Quality →",
  ...
)
```

Arguments

quality	Numeric vector of values between -1 and 1, denoting the quality of observations, where 0 is neutral.
amount	Numeric vector of values between 0 and 1, denoting the relative amount of information.
where	Location of legend, passed to `par(fig = where)`
n	Integer vector giving number of cells to plot in swatch for `quality` and `amount`.
Col	Function that takes vectors `amount` and `quality` and returns a vector of colours. QCol colours by data quality (concordance); QACol by quality and amount of information.
xlab	Character giving a label for the x axis.
ylab	Character giving a label for the y axis.
...	Additional parameters to `mtext()`.

Value

QACol() returns an RGB hex code for a colour, where lighter colours correspond to entries with a higher amount; unsaturated colours denote a neutral quality; and red/cyan colours denote low/high quality.

QCol() returns an RGB hex code for a colour, where darker, unsaturated colours denote a neutral quality; and red/cyan colours denote low/high quality. amount is ignored.

Author(s)

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

See Also

Other utility functions: ClusterStrings(), QuartetResolution(), WhenFirstHit()

Examples

```
amount <- runif(80, 0, 1)
quality <- runif(80, -1, 1)
plot(quality, amount, col = QACol(quality, amount), pch = 15)
abline(v = 0)
```

QuartetResolution *Relationship between four taxa*

Description

Relationship between four taxa

Usage

```
QuartetResolution(trees, tips)
```

Arguments

trees	A list of trees of class phylo, or a multiPhylo object.
tips	Vector specifying four tips whose relationship should be reported, in a format accepted by KeepTip().

Value

A vector specifying an integer, for each tree, which of tips[-1] is most closely related to tips[1].

See Also

Other utility functions: ClusterStrings(), QACol(), WhenFirstHit()

Examples

```
trees <- inapplicable.trees[["Vinther2008"]]
tips <- c("Lingula", "Halkieria", "Wiwaxia", "Acaenoplax")
QuartetResolution(trees, tips)
```

`RandomMorphyTree` *Random postorder tree*

Description

Random postorder tree

Usage

```
RandomMorphyTree(nTip)
```

Arguments

`nTip`	Integer specifying the number of tips to include in the tree (minimum 2).

Value

A list with three elements, each a vector of integers, respectively containing:

- The parent of each tip and node, in order
- The left child of each node
- The right child of each node.

See Also

Other tree generation functions: `AdditionTree()`

`RandomTreeScore` *Parsimony score of random postorder tree*

Description

Parsimony score of random postorder tree

Usage

```
RandomTreeScore(morphyObj)
```

Arguments

`morphyObj`	Object of class `morphy`, perhaps created with `PhyDat2Morphy()`.

Value

RandomTreeScore() returns the parsimony score of a random tree for the given Morphy object.

Examples

```
tokens <- matrix(c(
  0, "-", "-", 1, 1, 2,
  0, 1, 0, 1, 2, 2,
  0, "-", "-", 0, 0, 0), byrow = TRUE, nrow = 3L,
  dimnames = list(letters[1:3], NULL))
pd <- TreeTools::MatrixToPhyDat(tokens)
morphyObj <- PhyDat2Morphy(pd)

RandomTreeScore(morphyObj)

morphyObj <- UnloadMorphy(morphyObj)
```

RearrangeEdges *Rearrange edges of a phylogenetic tree*

Description

RearrangeEdges() performs the specified edge rearrangement on a matrix that corresponds to the edges of a phylogenetic tree, returning the score of the new tree. Will generally be called from within a tree search function.

Usage

```
RearrangeEdges(
  parent,
  child,
  dataset,
  TreeScorer = MorphyLength,
  EdgeSwapper,
  scoreToBeat = TreeScorer(parent, child, dataset, ...),
  iter = "?",
  hits = 0L,
  verbosity = 0L,
  ...
)
```

Arguments

parent	Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 1]
child	Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 2].

dataset	Third argument to pass to TreeScorer.
TreeScorer	function to score a given tree. The function will be passed three parameters, corresponding to the parent and child entries of a tree's edge list, and a dataset.
EdgeSwapper	a function that rearranges a parent and child vector, and returns a list with modified vectors; for example SPRSwap().
scoreToBeat	Double giving score of input tree.
iter	iteration number of calling function, for reporting to user only.
hits	Integer giving number of times the input tree has already been hit.
verbosity	Numeric specifying level of detail to display in console: larger numbers provide more verbose feedback to the user.
...	further arguments to pass to TreeScorer(), e.g. dataset = .

Details

RearrangeTree() performs one tree rearrangement of a specified type, and returns the score of the tree (with the given dataset). It also reports the number of times that this score was hit in the current function call.

Value

This function returns a list with two to four elements, corresponding to a binary tree: - 1. Integer vector listing the parent node of each edge; - 2. Integer vector listing the child node of each edge; - 3. Score of the tree; - 4. Number of times that score has been hit.

Author(s)

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

Examples

```
data("Lobo", package="TreeTools")
tree <- TreeTools::NJTree(Lobo.phy)
edge <- tree$edge
parent <- edge[, 1]
child <- edge[, 2]
dataset <- PhyDat2Morphy(Lobo.phy)
RearrangeEdges(parent, child, dataset, EdgeSwapper = RootedNNISwap)
# Remember to free memory:
dataset <- UnloadMorphy(dataset)
```

referenceTree *Tree topology for matrix simulation*

Description

The tree topology used to generate the matrices in congreveLamsdellMatrices

Usage

```
referenceTree
```

Format

A single phylogenetic tree saved as an object of class phylo

Source

Congreve & Lamsdell (2016); doi:10.1111/pala.12236

References

Congreve CR, Lamsdell JC (2016). "Implied weighting and its utility in palaeontological datasets: a study using modelled phylogenetic matrices." *Palaeontology*, **59**(3), 447–465. doi:10.1111/pala.12236. Congreve CR, Lamsdell JC (2016). "Data from: Implied weighting and its utility in palaeontological datasets: a study using modelled phylogenetic matrices." *Dryad Digital Repository*, doi:10.5061/dryad.7dq0j. doi:10.5061/dryad.7dq0j.

Examples

```
data(referenceTree)
plot(referenceTree)
```

SingleCharMorphy *Morphy object from single character*

Description

Morphy object from single character

Usage

```
SingleCharMorphy(char, gap = "inapp")
```

Arguments

char	State of each character at each tip in turn, in a format that will be converted to a character string by paste0(char, ";", collapse="").
gap	An unambiguous abbreviation of inapplicable, ambiguous (= missing), or extra state, specifying how gaps will be handled.

Value

A pointer to an object of class morphyObj. Don't forget to unload it when you've finished with it.

Author(s)

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

See Also

Score a tree: MorphyTreeLength()

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), MorphyWeights(), PhyDat2Morphy(), UnloadMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

Examples

```
morphyObj <- SingleCharMorphy("-0-0", gap = "Extra")
RandomTreeScore(morphyObj)
morphyObj <- UnloadMorphy(morphyObj)
```

SiteConcordance *Concordance factors*

Description

Concordance measures the strength of support that characters in a dataset present for each split (=edge/branch) in a tree (Minh et al. 2020; Smith 2026).

Usage

```
ClusteringConcordance(tree, dataset, return = "edge", normalize = TRUE)

MutualClusteringConcordance(tree, dataset)

QuartetConcordance(tree, dataset = NULL, weight = TRUE, return = "edge")
```

```
PhylogeneticConcordance(tree, dataset)

SharedPhylogeneticConcordance(tree, dataset)
```

Arguments

tree	A tree of class phylo.
dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
return	Character specifying the summary to return. Options are: • "edge": average concordance of each tree split across all characters; • "char": concordance of each character averaged over all splits, weighted by each character's information content; • "tree": an overall tree-level concordance score; • "all": a full array of MI components and normalized values for every split–character pair. Matching is case-insensitive and partial.
normalize	Controls how the *expected* mutual information (the zero point of the scale) is determined. • FALSE: no chance correction; MI is scaled only by its maximum. • TRUE: subtract the analytical expected MI for random association. • <integer>: subtract an empirical expected MI estimated from that number of random trees. In all cases, 1 corresponds to the maximal attainable MI for the pair (hBest), and 0 corresponds to the chosen expectation.
weight	Logical specifying whether to weight sites according to the number of quartets they are decisive for.

Details

ClusteringConcordance() measures how well each tree split reflects the grouping structure implied by each character (Smith 2026). Characters and splits are treated as clusterings of taxa, and their agreement is quantified using mutual information (MI). All reported values are scaled so that 1 corresponds to the maximum possible mutual information for each split–character pair (hBest).

The normalize argument specifies how the zero point is defined.

- If normalize = FALSE, zero corresponds to *zero* MI, without correcting for the positive bias that arises because MI is rarely exactly zero in finite samples.
- If normalize = TRUE, the expected MI is computed using an analytical approximation based on the distribution of character tokens. This is fast and generally accurate for large trees (~200+ taxa), but does not account for correlation between splits.
- If normalize is a positive integer n, the expected MI is estimated empirically by fitting each character to n uniformly random trees and averaging the resulting MI values. This Monte Carlo approach provides a more accurate baseline for small trees, for which the analytical approximation is biased. Monte Carlo standard errors are returned.

`MutualClusteringConcordance()` provides a character-wise summary that emphasises each character's best-matching split(s). It treats each character as a simple tree and computes the mutual clustering information between this character-tree and the supplied phylogeny. High values identify characters whose signal is well represented anywhere in the tree, even if concentrated on a single edge.

`QuartetConcordance()` is the proportion of quartets (sets of four leaves) that are decisive for a split which are also concordant with it For example, a quartet with the characters `0 0 0 1` is not decisive, as all relationships between those leaves are equally parsimonious. But a quartet with characters `0 0 1 1` is decisive, and is concordant with any tree that groups the first two leaves together to the exclusion of the second. In contrast to the site concordance factor (Minh et al. 2020), `QuartetConcordance()` considers all quartets that are decisive for a branch. Doing so circumvents the criticisms of Goloboff (2024).

By default, the reported value weights each site by the number of quartets it is decisive for. This value can be interpreted as the proportion of all decisive quartets that are concordant with a split. If `weight = FALSE`, the reported value is the mean of the concordance value for each site. Consider a split associated with two sites: one that is concordant with 25% of 96 decisive quartets, and a second that is concordant with 75% of 4 decisive quartets. If `weight = TRUE`, the split concordance will be 24 + 3 / 96 + 4 = 27%. If `weight = FALSE`, the split concordance will be mean(75%, 25%) = 50%.

`QuartetConcordance()` is computed exactly, using all quartets, rather than a random subsample (cf. Minh et al. 2020). Ambiguous and inapplicable tokens are treated as containing no grouping information (i.e. `(02)` or `-` are each treated as ?).

`PhylogeneticConcordance()` treats each character in `dataset` as a phylogenetic hypothesis and measures the extent to which it supports the splits of `tree`. Each character is first interpreted as a tree (or set of trees) in which taxa sharing the same token form a clade. Only splits for which the character contains at least four relevant taxa can contribute information.

For each split, the function identifies which characters could potentially support that split (i.e. those for which the induced subtrees contain informative structure), and among these, which characters are actually compatible with the split. The concordance value for each split is the proportion of informative characters that support it. A value of 1 indicates that all characters informative for that subset of taxa support the split; a value of 0 indicates that none do. Characters that contain only ambiguous or uninformative states for the relevant taxa do not affect the result.

`SharedPhylogeneticConcordance()` treats each character as a simple tree. Each token in the character corresponds to a node whose pendant edges are the taxa with that token. The Shared Phylogenetic Concordance for each character in `dataset` is then the Shared Phylogenetic Information (Smith 2020) of this tree and `tree`.

Value

`ClusteringConcordance(return = "all")` returns a 3D array where each slice corresponds to a character (site), each column to a tree split, and each row to a different information measure. The `normalized` row gives the normalized mutual information between each split-character pair, scaled so that 1.0 corresponds to `hBest` (the theoretical maximum mutual information, being the minimum of `hSplit` and `hChar`) and 0.0 corresponds to `miRand` (the expected mutual information under random association). `hSplit` gives the entropy (information content) of each split's bipartition; `hChar` gives the entropy of each character's state distribution; `hJoint` gives the joint entropy of the split-character confusion matrix; `mi` gives the raw mutual information; and `n` records the number of

informative observations. Negative normalized values indicate observed mutual information below random expectation. NA is returned when hBest = 0 (no information potential).

ClusteringConcordance(return = "edge") returns a vector where each element corresponds to a split (an edge of the tree) and gives the normalized mutual information between that split and the character data, averaged across all characters. When normalize = TRUE (default), values are scaled relative to random expectation; when FALSE, raw mutual information normalized by hBest is returned.

ClusteringConcordance(return = "char") returns a vector where each element corresponds to a character (site) and gives the entropy-weighted average normalized mutual information between that character and all tree splits. Characters with higher information content receive proportionally more weight from splits that can potentially convey more information about them.

ClusteringConcordance(return = "tree") returns a single value representing the overall concordance between the tree topology and the character data. This averages the fit of the best-matching split for each character. This is included for completeness, though it is not clear that this is a useful or meaningful measure.

MutualClusteringConcordance() returns the mutual clustering concordance of each character in dataset with tree. The attribute weighted.mean gives the mean value, weighted by the information content of each character.

QuartetConcordance(return = "edge") returns a numeric vector giving the concordance index at each split across all sites; names specify the number of each corresponding split in tree.

QuartetConcordance(return = "char") returns a numeric vector giving the concordance index calculated at each site, averaged across all splits.

PhylogeneticConcordance() returns a numeric vector giving the phylogenetic information of each split in tree, named according to the split's internal numbering.

SharedPhylogeneticConcordance() returns the shared phylogenetic concordance of each character in dataset with tree. The attribute weighted.mean gives the mean value, weighted by the information content of each character.

Author(s)

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

References

Goloboff PA (2024). "Measuring Degree of Group Support." In *Phylogenetic Analysis of Morphological Data*, volume 2, 119–183. CRC Press. doi:10.1201/97803678234128.

Minh BQ, Hahn MW, Lanfear R (2020). "New methods to calculate concordance factors for phylogenomic datasets." *Molecular Biology and Evolution*, **37**(9), 2727–2733. doi:10.1093/molbev/msaa106.

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 (2026). "Which characters support which clades? Exploring the distribution of phylogenetic signal using mutual information." *Systematic Biology*, Under review.

See Also

- ConcordanceTable(): Visualize concordance.
- Consistency(): Measure consistency of characters with dataset.

Other split support functions: ConcordanceTable(), JackLabels(), Jackknife(), MaximizeParsimony(), MostContradictedFreq(), PresCont()

Examples

```
data("congreveLamsdellMatrices", package = "TreeSearch")
dataset <- congreveLamsdellMatrices[[1]][, 1:20]
tree <- TreeSearch::referenceTree

cc <- ClusteringConcordance(tree, dataset)
mcc <- MutualClusteringConcordance(tree, dataset)

qc <- QuartetConcordance(tree, dataset)

pc <- PhylogeneticConcordance(tree, dataset)
spc <- SharedPhylogeneticConcordance(tree, dataset)

plot(tree)
TreeTools::LabelSplits(tree, signif(qc, 3), cex = 0.8)
plot(tree)
TreeTools::LabelSplits(tree, signif(cc, 3), cex = 0.8)

# Write concordance factors to file
labels <- paste0(cc, "/", qc, "/", pc) # "/" is a valid delimiter
# Identify the node that corresponds to each label
whichNode <- match(TreeTools::NTip(tree) + 1:tree$Nnode, names(qc))

# The contents of tree$node.label will be written at each node
tree$node.label <- labels[whichNode]

ape::write.tree(tree) # or write.nexus(tree, file = "mytree.nex")

#' # Display correlation between concordance factors
pairs(cbind(cc, mcc, qc, pc, spc), asp = 1)
data(congreveLamsdellMatrices)
myMatrix <- congreveLamsdellMatrices[[10]]
ClusteringConcordance(TreeTools::NJTree(myMatrix), myMatrix)
```

SPR *Subtree pruning and rearrangement (SPR)*

Description

Perform one SPR rearrangement on a tree

Usage

```
SPR(tree, edgeToBreak = NULL, mergeEdge = NULL)

SPRMoves(tree, edgeToBreak = integer(0))

## S3 method for class 'phylo'
SPRMoves(tree, edgeToBreak = integer(0))

## S3 method for class 'matrix'
SPRMoves(tree, edgeToBreak = integer(0))

SPRSwap(
  parent,
  child,
  nEdge = length(parent),
  nNode = nEdge/2L,
  edgeToBreak = NULL,
  mergeEdge = NULL
)

RootedSPR(tree, edgeToBreak = NULL, mergeEdge = NULL)

RootedSPRSwap(
  parent,
  child,
  nEdge = length(parent),
  nNode = nEdge/2L,
  edgeToBreak = NULL,
  mergeEdge = NULL
)
```

Arguments

tree	A bifurcating tree of class phylo, with all nodes resolved;
edgeToBreak	the index of an edge to bisect, generated randomly if not specified.
mergeEdge	the index of an edge on which to merge the broken edge.
parent	Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. `tree[["edge"]][, 1]`
child	Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. `tree[["edge"]][, 2]`.
nEdge	(optional) integer specifying the number of edges of a tree of class phylo, i.e. `dim(tree$edge)[1]`
nNode	(optional) Number of nodes.

Details

Equivalent to kSPR() in the **phangorn** package, but faster. Note that rearrangements that only change the position of the root WILL be returned by SPR. If the position of the root is irrelevant (as in Fitch parsimony, for example) then this function will occasionally return a functionally equivalent topology. RootIrrelevantSPR will search tree space more efficiently in these cases. Branch lengths are not (yet) supported.

All nodes in a tree must be bifurcating; ape::collapse.singles and ape::multi2di may help.

Value

This function returns a tree in phyDat format that has undergone one SPR iteration.

TBRMoves() returns a list of all trees one SPR move away from tree, with edges and nodes in preorder, rooted on the first-labelled tip.

a list containing two elements, corresponding in turn to the rearranged parent and child parameters

a list containing two elements, corresponding in turn to the rearranged parent and child parameters

Functions

- SPRSwap(): faster version that takes and returns parent and child parameters
- RootedSPR(): Perform SPR rearrangement, retaining position of root
- RootedSPRSwap(): faster version that takes and returns parent and child parameters

Author(s)

Martin R. Smith

References

The SPR algorithm is summarized in Felsenstein J (2004). *Inferring phylogenies*. Sinauer Associates, Sunderland, Massachusetts.

See Also

- RootedSPR(): useful when the position of the root node should be retained.

Other tree rearrangement functions: NNI(), TBR()

Examples

```
{
tree <- ape::rtree(20, br=FALSE)
SPR(tree)
}
```

`StepInformation` *Information content of a character known to contain* e *steps*

Description

`StepInformation()` calculates the phylogenetic information content of a character `char` when *e* extra steps are present, for all possible values of *e*.

Usage

```
StepInformation(char, ambiguousTokens = c("-", "?"))
```

Arguments

`char`	Vector of tokens listing states for the character in question.
`ambiguousTokens`	Vector specifying which tokens, if any, correspond to the ambiguous token (?).

Details

Calculates the number of trees consistent with the character having *e* extra steps, where *e* ranges from its minimum possible value (i.e. number of different tokens minus one) to its maximum.

Value

`StepInformation()` returns a numeric vector detailing the amount of phylogenetic information (in bits) associated with the character when 0, 1, 2… extra steps are present. The vector is named with the total number of steps associated with each entry in the vector: for example, a character with three observed tokens must exhibit two steps, so the first entry (zero extra steps) is named 2 (two steps observed).

Author(s)

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

See Also

Other profile parsimony functions: `Carter1()`, `PrepareDataProfile()`, `WithOneExtraStep()`, `profiles`

Examples

```
character <- rep(c(0:3, "?", "-"), c(8, 5, 1, 1, 2, 2))
StepInformation(character)
```

summary.morphyPtr *Details the attributes of a morphy object*

Description

Details the attributes of a morphy object

Usage

```
## S3 method for class 'morphyPtr'
summary(object, ...)
```

Arguments

object	A Morphy object
...	any other parameters...

Value

A list detailing the number of taxa, internal nodes, and characters and their weights.

Author(s)

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

See Also

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), MorphyWeights(), PhyDat2Morphy(), SingleCharMorphy(), UnloadMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip()

TaxonInfluence *Rank taxa by their influence on phylogenetic results*

Description

TaxonInfluence() ranks taxa according to their influence on the most parsimonious topology.

Usage

```
TaxonInfluence(
  dataset,
  tree = NULL,
  Distance = ClusteringInfoDistance,
  calcWeighted = TRUE,
  savePath = NULL,
  useCache = FALSE,
  verbosity = 3L,
  ...
)
```

Arguments

dataset	A phylogenetic data matrix of **phangorn** class phyDat, whose names correspond to the labels of any accompanying tree. Perhaps load into R using ReadAsPhyDat(). Additive (ordered) characters can be handled using Decompose().
tree	Optimal tree or summary tree (of class "phylo") or list of trees (of class "list" or "multiPhylo") against which results should be evaluated. If NULL, an optimal tree will be sought using parsimony search with the parameters provided in ....
Distance	Function to calculate tree distance; default: ClusteringInfoDistance().
calcWeighted	Logical specifying whether to compute the distance-weighted mean value.
savePath	Character giving prefix of path to which reduced trees will be saved (with write.nexus()). File names will follow the pattern paste0(savePath, droppedTaxonName, ".nex"); savePath should thus contain a trailing / if writing to a directory, which will be created if it does not exist. Special characters will be removed from leaf labels when creating the file path (using path_sanitize()). If NULL, computed trees will not be saved.
useCache	Logical vector; if TRUE, previous tree search results will be loaded from the location given by savePath, instead of running a fresh search with the specified dataset and parameters.
verbosity, ...	Parameters for MaximizeParsimony(). Tree search will be conducted using tree as a starting tree.

Details

TaxonInfluence() follows the approach of Mariadassou et al. (2012) in repeating tree search whilst leaving each taxon in turn out of the analysis, and measuring the distance of reconstructed trees from the optimal tree obtained when all taxa are included in phylogenetic inference.

As Denton and Goolsby (2018) emphasize, the Robinson–Foulds distance is unsuitable for this purpose; this function allows the user to specify a preferred tree distance measure, defaulting to the clustering information distance (Smith 2020). Because optimal parsimony trees are not equiprobable, taxon influence is ranked based on the maximum and minimum tree-to-tree distances between optimal trees.

Value

`TaxonInfluence()` returns a matrix listing the phylogenetic influence of each taxon, measured in the units of the chosen tree distance metric (default = bits). Columns denote taxa; rows denote the maximum, distance-weighted mean, and minimum distance between optimal tree sets.

Distance-weighted mean

Sets of equally parsimonious trees are not statistical samples of tree space, but are biased towards areas of uncertainty. It is possible that a set of trees contains all possible resolutions of a particular clade, and a single other topology in which that clade does not exist – essentially two distinct solutions, one (*a*) which could be summarised with a summary tree that contains a polytomy, and another (*b*) which could be summarized by a perfectly resolved tree. Neither of these scenarios is preferable under the principles of parsimony; but summary statistics (e.g. mean, median) will be strongly influenced by the many trees in group *a*, thus underplaying the existence of solution *b*.

`TaxonInfluence()` uses an *ad hoc* method to produce summary statistics after weighting for trees' distance from other trees. Trees that have few close neighbours contribute more to the weighted mean, thus reducing the influence of many trees that differ only in small details. This distance-weighted mean is thus less prone to bias than a simple mean – it is no more statistically valid, but (potentially) provides a more representative summary of comparisons between sets of trees.

Author(s)

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

References

Denton JS, Goolsby EW (2018). "Measuring Inferential Importance of Taxa Using Taxon Influence Indices." *Ecology and Evolution*, **8**(9), 4484–4494. doi:10.1002/ece3.3941.

Mariadassou M, Bar-Hen A, Kishino H (2012). "Taxon Influence Index: Assessing Taxon-Induced Incongruities in Phylogenetic Inference." *Systematic Biology*, **61**(2), 337–345. doi:10.1093/sysbio/syr129.

Smith MR (2020). "Information Theoretic Generalized Robinson-Foulds Metrics for Comparing Phylogenetic Trees." *Bioinformatics*, **36**(20), 5007–5013. doi:10.1093/bioinformatics/btaa614.

See Also

Other tree scoring: `CharacterLength()`, `ExpectedLength()`, `IWScore()`, `LengthAdded()`, `MinimumLength()`, `MorphyTreeLength()`

Examples

```
#' # Load data for analysis in R
library("TreeTools")
data("congreveLamsdellMatrices", package = "TreeSearch")

# Small dataset for demonstration purposes
dataset <- congreveLamsdellMatrices[[42]][1:8, ]
```

```
bestTree <- MaximizeParsimony(dataset, verbosity = 0)[[1]]

# Calculate tip influence
influence <- TaxonInfluence(dataset, ratchIt = 0, startIt = 0, verbos = 0)

# Colour tip labels according to their influence
upperBound <- 2 * TreeDist::ClusteringEntropy(
  PectinateTree(NTip(dataset) - 1))
nBin <- 128
bin <- cut(
  influence["max", ],
  breaks = seq(0, upperBound, length.out = nBin),
  include.lowest = TRUE
)
palette <- hcl.colors(nBin, "inferno")

plot(bestTree, tip.color = palette[bin])
PlotTools::SpectrumLegend(
  "bottomleft",
  palette = palette,
  title = "Tip influence / bits",
  legend = signif(seq(upperBound, 0, length.out = 4), 3),
  bty = "n"
)
```

TBR *Tree bisection and reconnection (TBR)*

Description

TBR performs a single random TBR iteration.

Usage

```
TBR(tree, edgeToBreak = NULL, mergeEdges = NULL)

TBRMoves(tree, edgeToBreak = integer(0))

## S3 method for class 'phylo'
TBRMoves(tree, edgeToBreak = integer(0))

## S3 method for class 'matrix'
TBRMoves(tree, edgeToBreak = integer(0))

TBRSwap(
  parent,
  child,
  nEdge = length(parent),
  edgeToBreak = NULL,
```

```
  mergeEdges = NULL
)

RootedTBR(tree, edgeToBreak = NULL, mergeEdges = NULL)

RootedTBRSwap(
  parent,
  child,
  nEdge = length(parent),
  edgeToBreak = NULL,
  mergeEdges = NULL
)
```

Arguments

tree	A bifurcating tree of class phylo, with all nodes resolved;
edgeToBreak	(optional) integer specifying the index of an edge to bisect/prune, generated randomly if not specified. Alternatively, set to -1 to return a complete list of all trees one step from the input tree.
mergeEdges	(optional) vector of length 1 or 2, listing edge(s) to be joined: In SPR, this is where the pruned subtree will be reconnected. In TBR, these edges will be reconnected (so must be on opposite sides of edgeToBreak); if only a single edge is specified, the second will be chosen at random
parent	Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 1]
child	Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. tree[["edge"]][, 2].
nEdge	(optional) Number of edges.

Details

Branch lengths are not (yet) supported.

All nodes in a tree must be bifurcating; ape::collapse.singles and ape::multi2di may help.

Value

TBR() returns a tree in phyDat format that has undergone one TBR iteration.

TBRMoves() returns a multiPhylo object listing all trees one TBR move away from tree, with edges and nodes in preorder, rooted on the first-labelled tip.

TBRSwap() returns a list containing two elements corresponding to the rearranged parent and child parameters.

Functions

- TBRSwap(): faster version that takes and returns parent and child parameters
- RootedTBR(): Perform TBR rearrangement, retaining position of root
- RootedTBRSwap(): faster version that takes and returns parent and child parameters

Author(s)

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

References

The TBR algorithm is summarized in Felsenstein J (2004). *Inferring phylogenies*. Sinauer Associates, Sunderland, Massachusetts.

See Also

`RootedTBR()`: useful when the position of the root node should be retained.

Other tree rearrangement functions: `NNI()`, `SPR()`

Examples

```
library("ape")
tree <- rtree(20, br=NULL)
TBR(tree)
```

UnloadMorphy *Destroy a Morphy object*

Description

Destroys a previously-created Morphy object.

Usage

```
UnloadMorphy(morphyObj)
```

Arguments

`morphyObj`	Object of class `morphy`, perhaps created with `PhyDat2Morphy()`.

Details

Best practice is to call `morphyObj <- UnloadMorphy(morphyObj)` Failure to do so will cause a crash if `UnloadMorphy()` is called on an object that has already been destroyed

Value

Morphy error code, decipherable using `mpl_translate_error`

Author(s)

Martin R. Smith

See Also

Other Morphy API functions: GapHandler(), MorphyErrorCheck(), MorphyWeights(), PhyDat2Morphy(), SingleCharMorphy(), is.morphyPtr(), mpl_apply_tipdata(), mpl_attach_rawdata(), mpl_attach_symbols(), mpl_delete_Morphy(), mpl_first_down_recon(), mpl_first_up_recon(), mpl_get_charac_weight(), mpl_get_gaphandl(), mpl_get_num_charac(), mpl_get_num_internal_nodes(), mpl_get_numtaxa(), mpl_get_symbols(), mpl_init_Morphy(), mpl_new_Morphy(), mpl_second_down_recon(), mpl_second_up_recon(), mpl_set_charac_weight(), mpl_set_num_internal_nodes(), mpl_set_parsim_t(), mpl_translate_error(), mpl_update_lower_root(), mpl_update_tip(), summary.morphyPtr()

WhenFirstHit *When was a tree topology first hit?*

Description

Reports when each tree in a list was first found by tree search. This information is read from the `firstHit` attribute if present. If not, trees are taken to be listed in the order in which they were found, and named according to the search iteration in which they were first hit - the situation when trees found by MaximizeParsimony() are saved to file.

Usage

```
WhenFirstHit(trees)
```

Arguments

`trees`	A list of trees, or a `multiPhylo` object.

Value

`trees`, with a `firstHit` attribute listing the number of trees hit for the first time in each search iteration.

Author(s)

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

See Also

- MaximizeParsimony()

Other utility functions: ClusterStrings(), QACol(), QuartetResolution()

Examples

```
library("TreeTools", quietly = TRUE)
trees <- list(
   seed_00 = as.phylo(1, 8),
   ratch1_01 = as.phylo(2, 8),
   ratch1_02 = as.phylo(3, 8),
   ratch4_44 = as.phylo(4, 8),
   final_99 = as.phylo(5, 8)
)
attr(WhenFirstHit(trees), "firstHit")
```

WithOneExtraStep *Number of trees with one extra step*

Description

Number of trees with one extra step

Usage

```
WithOneExtraStep(...)
```

Arguments

`...`	Vector or series of integers specifying the number of leaves bearing each distinct non-ambiguous token.

See Also

Other profile parsimony functions: `Carter1()`, `PrepareDataProfile()`, `StepInformation()`, `profiles`

Examples

```
WithOneExtraStep(1, 2, 3)
```

Index

∗ **Morphy API functions**
GapHandler, 16
is.morphyPtr, 21
MorphyWeights, 41
PhyDat2Morphy, 46
SingleCharMorphy, 59
summary.morphyPtr, 68
UnloadMorphy, 73
∗ **custom search functions**
Jackknife, 24
MorphyBootstrap, 38
∗ **datasets**
congreveLamsdellMatrices, 12
inapplicable.datasets, 17
profiles, 53
referenceTree, 59
∗ **profile parsimony functions**
Carter1, 5
PrepareDataProfile, 50
profiles, 53
StepInformation, 67
WithOneExtraStep, 75
∗ **split support functions**
ConcordanceTable, 9
Jackknife, 24
JackLabels, 25
MaximizeParsimony, 29
MostContradictedFreq, 43
PresCont, 51
SiteConcordance, 60
∗ **tree generation functions**
AdditionTree, 3
RandomMorphyTree, 56
∗ **tree rearrangement functions**
NNI, 44
SPR, 64
TBR, 71
∗ **tree scoring**
CharacterLength, 6
ExpectedLength, 15
IWScore, 21
LengthAdded, 27
MinimumLength, 36
TaxonInfluence, 68
∗ **utility functions**
ClusterStrings, 8
QACol, 54
QuartetResolution, 55
WhenFirstHit, 74

addition tree, *30*
AdditionTree, 3
AdditionTree(), *56*
AllSPR, 4
ape::collapse.singles, *66*, *72*
ape::collapse.singles(), *44*
ape::consensus(), *41*
ape::multi2di, *66*, *72*
ape::multi2di(), *44*

Carter1, 5
Carter1(), *51*, *53*, *67*, *75*
CharacterLength, 6
CharacterLength(), *16*, *23*, *28*, *37*, *70*
ClusteringConcordance *(*SiteConcordance*)*, 60
ClusteringInfoDistance(), *69*
ClusterStrings, 8
ClusterStrings(), *55*, *74*
cNNI *(*NNI*)*, 44
ConcordanceTable, 9
ConcordanceTable(), *25, 26*, *34*, *43*, *52*, *64*
ConcordantInfo *(*ConcordantInformation*)*, 11
ConcordantInformation, 11
congreveLamsdellMatrices, 12, *59*
Consistency, 13
Consistency(), *64*
cSPR, 15

Decompose, *3*, *7*, *9*, *11*, *13*, *16*, *22*, *30*, *49*, *61*, *69*

EasyTrees *(*MaximizeParsimony*)*, 29
EasyTrees(), *23*, *34*
EasyTreesy *(*MaximizeParsimony*)*, 29
EdgeListSearch(), *25*, *41*
Evaluate *(*ConcordantInformation*)*, 11
ExpectedLength, 15
ExpectedLength(), *7*, *23*, *28*, *37*, *70*

FastCharacterLength *(*CharacterLength*)*, 6
Fitch *(*IWScore*)*, 21

GapHandler, 16
GapHandler(), *21*, *42*, *46*, *60*, *68*, *74*
graphical parameter, *49*

ImposeConstraint(), *4*, *31*
inapplicable.citations *(*inapplicable.datasets*)*, 17
inapplicable.datasets, 17
inapplicable.phyData *(*inapplicable.datasets*)*, 17
inapplicable.trees *(*inapplicable.datasets*)*, 17
is.morphyPtr, 21
is.morphyPtr(), *17*, *42*, *46*, *60*, *68*, *74*
IWScore, 21
IWScore(), *7*, *16*, *28*, *37*, *70*

Jackknife, 24
Jackknife(), *10*, *26*, *34*, *41*, *43*, *52*, *64*
JackLabels, 25
JackLabels(), *10*, *25*, *34*, *43*, *52*, *64*

KeepTip, *55*

LengthAdded, 27
LengthAdded(), *7*, *16*, *23*, *37*, *70*
Log2Carter1 *(*Carter1*)*, 5
LogCarter1 *(*Carter1*)*, 5

MaximizeParsimony, 29
MaximizeParsimony(), *10*, *23*, *25*, *26*, *43*, *52*, *64*, *69*, *74*
MaximumLength *(*MinimumLength*)*, 36
MaximumLength(), *13*
MinimumLength, 36
MinimumLength(), *7*, *16*, *23*, *28*, *70*
MinimumSteps *(*MinimumLength*)*, 36
MorphyBootstrap, 38
MorphyBootstrap(), *25*
MorphyErrorCheck(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
MorphyTreeLength(), *7*, *16*, *23*, *28*, *37*, *60*, *70*
MorphyWeights, 41
MorphyWeights(), *17*, *21*, *46*, *60*, *68*, *74*
MostContradictedFreq, 43
MostContradictedFreq(), *10*, *25*, *26*, *34*, *52*, *64*
mpl_apply_tipdata(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_attach_rawdata(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_attach_symbols(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_delete_Morphy(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_first_down_recon(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_first_up_recon(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_charac_weight(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_gaphandl(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_num_charac(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_num_internal_nodes(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_numtaxa(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_get_symbols(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_init_Morphy(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_new_Morphy(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_second_down_recon(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_second_up_recon(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_set_charac_weight(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_set_num_internal_nodes(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_set_parsim_t(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_translate_error, *73*
mpl_translate_error(), *17*, *21*, *42*, *46*, *60*, *68*, *74*

mpl_update_lower_root(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mpl_update_tip(), *17*, *21*, *42*, *46*, *60*, *68*, *74*
mtext(), *54*
MultiRatchet *(*MorphyBootstrap*)*, 38
MutualClusteringConcordance *(*SiteConcordance*)*, 60

NNI, 44
NNI(), *66*, *73*
NNISwap, *40*
NNISwap *(*NNI*)*, 44

paste0, *60*
path_sanitize(), *69*
PhyDat2Morphy, 46
PhyDat2Morphy(), *17*, *21*, *39*, *42*, *56*, *60*, *68*, *73*, *74*
phylo, *5*, *7*, *9*, *11*, *13*, *15*, *16*, *24*, *26*, *30*, *40*, *43*, *44*, *51*, *57*, *61*, *65*, *72*
PhylogeneticConcordance *(*SiteConcordance*)*, 60
PlotCharacter, 47
PolEscapa *(*LengthAdded*)*, 27
pratchet(), *41*
PrepareDataIW *(*PrepareDataProfile*)*, 50
PrepareDataProfile, 50
PrepareDataProfile(), *6*, *53*, *67*, *75*
PresCont, 51
PresCont(), *10*, *25*, *26*, *34*, *43*, *64*
profiles, *6*, *51*, 53, *67*, *75*

QACol, *9*, 54, *54*
QACol(), *8*, *55*, *74*
QALegend *(*QACol*)*, 54
QCol, *9*, *54*
QCol *(*QACol*)*, 54
QuartetConcordance *(*SiteConcordance*)*, 60
QuartetResolution, 55
QuartetResolution(), *8*, *55*, *74*

RandomMorphyTree, 56
RandomMorphyTree(), *4*
RandomTreeScore, 56
Ratchet *(*MorphyBootstrap*)*, 38
RatchetConsensus *(*MorphyBootstrap*)*, 38
read.nexus.data(), *17*
ReadAsPhyDat, *3*, *7*, *9*, *11*, *13*, *16*, *22*, *30*, *49*, *61*, *69*
RearrangeEdges, 57
referenceTree, *12*, 59
Resample *(*MaximizeParsimony*)*, 29
Resample(), *25*, *26*
RootedNNI *(*NNI*)*, 44
RootedNNISwap, *40*
RootedNNISwap *(*NNI*)*, 44
RootedSPR *(*SPR*)*, 64
RootedSPR(), *66*
RootedSPRSwap *(*SPR*)*, 64
RootedTBR *(*TBR*)*, 71
RootedTBR(), *73*
RootedTBRSwap, *40*
RootedTBRSwap *(*TBR*)*, 71

SetMorphyWeights *(*MorphyWeights*)*, 41
SharedPhylogeneticConcordance *(*SiteConcordance*)*, 60
SingleCharMorphy, 59
SingleCharMorphy(), *17*, *21*, *42*, *46*, *68*, *74*
SiteConcordance, *10*, *25*, *26*, *34*, *43*, *52*, 60
SiteConcordance(), *10*, *33*
SplitFrequency, *52*
SPR, 64
SPR(), *45*, *73*
SPRMoves *(*SPR*)*, 64
SPRSwap *(*SPR*)*, 64
SPRSwap(), *24*, *39*, *58*
StepInformation, 67
StepInformation(), *6*, *51*, *53*, *75*
SuccessiveApproximations(), *25*, *41*
summary.morphyPtr, 68
summary.morphyPtr(), *17*, *21*, *42*, *46*, *60*, *74*

TaxonInfluence, 68
TaxonInfluence(), *7*, *16*, *23*, *28*, *37*
TBR, 71
TBR(), *45*, *66*
TBRMoves *(*TBR*)*, 71
TBRSwap *(*TBR*)*, 71
TreeLength *(*IWScore*)*, 21
TreeLength(), *14*
TreeSearch(), *23*
TreeTools::ConsensusWithout(), *41*
TreeTools::StringToPhyDat(), *36*

UnloadMorphy, 73
UnloadMorphy(), *17*, *21*, *42*, *46*, *60*, *68*

WhenFirstHit, 74

WhenFirstHit(), *8*, *55*
WithOneExtraStep, 75
WithOneExtraStep(), *6*, *51*, *53*, *67*
write.nexus(), *69*