Package 'Claddis'

February 12, 2019

Type Package

Title Measuring Morphological Diversity and Evolutionary Tempo

Version 0.3.0

Date 2019-02-12

Maintainer Graeme T. Lloyd <graemetlloyd@gmail.com>

Depends ape, phytools, strap

Imports gdata, graphics, grDevices, stats, utils

Suggests rgl

Description Measures morphological diversity from discrete character data and estimates evolutionary tempo on phylogenetic trees. Imports morphological data from #NEXUS (Maddison et al. (1997) <doi:10.1093/sysbio/46.4.590>) format with ReadMorphNexus(), and writes to both #NEXUS and TNT format (Goloboff et al. (2008) <doi:10.1111/j.1096-0031.2008.00217.x>). Main functions are DiscreteCharacterRate(), which implements likelihood ratio tests for discrete character rates introduced across Lloyd et al. (2012) <doi:10.1111/j.1558-5646.2011.01460.x>, Brusatte et al. (2014) <doi:10.1016/j.cub.2014.08.034>, Close et al. (2015) <doi:10.1016/j.cub.2015.06.047>, and Lloyd (2016) <doi:10.1111/bij.12746>, and MorphDistMatrix(), which implements multiple discrete character distance metrics from Gower (1971) <doi:10.2307/2528823>, Wills (1998) <doi:10.1006/bijl.1998.0255>, Lloyd (2016) <doi:10.1111/bij.12746>, and Hopkins and St John (2018) <doi:10.1098/rspb.2018.1784>. Multiple functions implement various morphospace plots: ChronoPhyloMorphospacePlot() implements Sakamoto and Ruta (2012) <doi:10.1371/journal.pone.0039752>, MorphospacePlot() implements Wills et al. (1994) <doi:10.1017/S009483730001263X>, PlotCharacterChanges() implements Wang and Lloyd (2016) <doi:10.1098/rspb.2016.0214>, and StackPlot() implements Foote (1993) <doi:10.1017/S0094837300015864>. Other functions include SafeTaxonomicReduction(), which implements Wilkinson (1995) <doi:10.1093/sysbio/44.4.501>, and DolloSCM() implements the Dollo stochastic character mapping of Tarver et al. (2018) <doi:10.1093/gbe/evy096>.

Encoding UTF-8

License GPL (>= 2)

LazyData yes

ByteCompile yes

RoxygenNote 6.1.1

NeedsCompilation no

Author Graeme T. Lloyd [aut, cre, cph],
Thomas Guillerme [aut, cph],
Emma Sherratt [aut, cph],
Steve C. Wang [aut, cph]

Repository CRAN

Date/Publication 2019-02-12 13:19:36 UTC

R topics documented:

`Claddis-package` *Measuring Morphological Diversity and Evolutionary Tempo*

Description

Measures morphological diversity from discrete character data and estimates evolutionary tempo on phylogenetic trees.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Lloyd, G. T., 2016. Estimating morphological diversity and tempo with discrete character-taxon matrices: implementation, challenges, progress, and future directions. Biological Journal of the Linnean Society, 118, 131-151.

Examples

```
# Get morphological distances for Michaux (1989) data set:
distances <- MorphDistMatrix(Michaux1989)

# Show distances:
distances
```

`AncStateEstMatrix` *Ancestral Character State Estimation*

Description

Given a tree and a cladistic matrix uses likelihood to estimate the ancestral states for every character.

Usage

```
AncStateEstMatrix(CladisticMatrix, Tree, EstimateAllNodes = FALSE,
  EstimateTipValues = FALSE, InapplicablesAsMissing = FALSE,
  PolymorphismBehaviour = "equalp", UncertaintyBehaviour = "equalp",
  Threshold = 0.01)
```

Arguments

`CladisticMatrix`
: A character-taxon matrix in the format imported by ReadMorphNexus.

`Tree`
: A tree (phylo object) with branch lengths that represents the relationships of the taxa in `CladisticMatrix`.

`EstimateAllNodes`
: Logical that allows the user to make estimates for all ancestral values. The default (`FALSE`) will only make estimates for nodes that link coded terminals (recommended).

`EstimateTipValues`
: Logical that allows the user to make estimates for tip values. The default (`FALSE`) will only makes estimates for internal nodes (recommended).

`InapplicablesAsMissing`
: Logical that decides whether or not to treat inapplicables as missing (TRUE) or not (FALSE, the default and recommended option).

`PolymorphismBehaviour`
: One of either "equalp" or "treatasmissing".

`UncertaintyBehaviour`
: One of either "equalp" or "treatasmissing".

`Threshold`
: The threshold value to use when collapsing marginal likelihoods to discrete state(s).

Details

Uses either the rerootingMethod (Yang et al. 1995) as implemented in the phytools package (discrete characters) or the ace function in the ape package (continuous characters) to make ancestral state estimates. For discrete characters these are collapsed to the most likely state (or states, given equal likelihoods or likelihood within a defined threshold value). In the latter case the resulting states are represented as an uncertainty (i.e., states separated by a slash, e.g., 0/1. This is the method used by Brusatte et al. (2014).

Value

`anc.lik.matrix`
: A matrix of nodes (hypothetical ancestors; rows) against characters (columns) listing the reconstructed ancestral states.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com> and Thomas Guillerme <guillert@tcd.ie>

References

Brusatte, S. L., Lloyd, G. T., Wang, S. C. and Norell, M. A., 2014. Gradual assembly of avian body plan culminated in rapid rates of evolution across dinosaur-bird transition. Current Biology, 24, 2386-2392.

Yang, Z., Kumar, S. and Nei, M., 1995. A new method of inference of ancestral nucleotide and amino acid sequences. Genetics, 141, 1641-1650.

Examples

```
# Set random seed:
set.seed(4)

# Generate a random tree for the Day data set:
tree <- rtree(n = nrow(Day2016$Matrix_1$Matrix))

# Update taxon names to match those in the data matrix:
tree$tip.label <- rownames(Day2016$Matrix_1$Matrix)

# Set root time by making youngest taxon extant:
tree$root.time <- max(diag(vcv(tree)))

# Use Day matrix as cladistic matrix:
CladisticMatrix <- Day2016

# Prune most characters out to make example run fast:
CladisticMatrix <- MatrixPruner(CladisticMatrix,
  characters2prune = c(2:3, 5:37))

# Estimate ancestral states:
AncStateEstMatrix(CladisticMatrix = CladisticMatrix,
  Tree = tree)
```

ChangesInBins *Counts the changes in a series of time bins*

Description

Given a vector of dates for a series of time bins and another for the times when a character change occurred will return the total number of changes in each bin.

Usage

```
ChangesInBins(change.times, time.bins)
```

Arguments

change.times	A vector of ages in millions of years at which character changes are hypothesised to have occurred.
time.bins	A vector of ages in millions of years of time bin boundaries in old-to-young order.

Details

Calculates the total number of evolutionary changes in a series of time bins. This is intended as an internal function for rate calculations, but could be used for other purposes (e.g., counting any point events in a series of time bins).

Value

A vector giving the number of changes for each time bin. Names indicate the maximum and minimum (bottom and top) values for each time bin.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random dataset of 100 changes:
change.times <- runif(100, 0, 100)

# Create time bins:
time.bins <- seq(100, 0, length.out=11)

# Get N changes for each bin:
ChangesInBins(change.times, time.bins)
```

ChronoPhyloMorphospacePlot

Chronophylomorphospace Plot

Description

Plots a 3D chronophylomorphospace.

Usage

```
ChronoPhyloMorphospacePlot(pcoa_data, x_axis = 1, y_axis = 2,
  shadow = TRUE)
```

Arguments

pcoa_data	Principal coordinate data in the format output by MorphMatrix2PCoA that includes a tree and ancestral states.
x_axis	Which ordination axis to plot as the x-axis.
y_axis	Which ordination axis to plot as the y-axis.
shadow	Whether or not to plot a shadow (2D plot) on the bottom face of the 3D plot (defaults to TRUE).

Details

Creates a movable three-dimensional (two ordination axes plus time) plot of a phylomorphospace.

Intended to mimic the data visualisation of Sakamoto and Ruta (2012; their Video S1).

Author(s)

Emma Sherratt <emma.sherratt@gmail.com> and Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Sakamoto, M. and Ruta, M. 2012. Convergence and divergence in the evolution of cat skulls: temporal and spatial patterns of morphological diversity. PLoS ONE, 7, e39752.

Examples

```
# Set random seed:
set.seed(4)

# Generate a random tree for the Michaux 1989 data set:
Tree <- rtree(nrow(Michaux1989$Matrix_1$Matrix))

# Set root time so latest tip terminates at the present:
Tree$root.time <- max(diag(vcv(Tree)))

# Add taxon names to the tree:
Tree$tip.label <- rownames(Michaux1989$Matrix_1$Matrix)

# Perform a phylogenetic Principal Coordinates Analysis:
pcoa_data <- MorphMatrix2PCoA(Michaux1989, Tree = Tree)

# Plot a chronophylomorphospace:
ChronoPhyloMorphospacePlot(pcoa_data)
```

CompactifyMatrix *Collapses matrix to unique character state distributions*

Description

Collapses a cladistic matrix to just unique character state distributions.

Usage

```
CompactifyMatrix(CladisticMatrix)
```

Arguments

`CladisticMatrix`
: The cladistic matrix in the format imported by ReadMorphNexus.

Details

Important: not recommended for general use.

This function is intended to make a matrix with redundant character state distributions smaller by collapsing these to single characters and upweighting them accordingly. It is intended purely for use with MRP matrices, but may have some very restricted uses elsewhere.

The function also deletes any characters weighted zero from the matrix.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

SafeTaxonomicReduction and MatrixPruner

Examples

```
# Examine the matrix pre-compactification:
Michaux1989$Matrix_1$Matrix

# Examine the weights pre-compactification:
Michaux1989$Matrix_1$Weights

# Compactify the matrix:
Michaux1989compact <- CompactifyMatrix(Michaux1989)

# Examine the matrix post-compactification:
Michaux1989compact$Matrix_1$Matrix

# Examine the weights post-compactification:
Michaux1989compact$Matrix_1$Weights
```

CorrectRootTime *Corrects root.time after taxa have been pruned from a tree*

Description

Corrects root.time after taxa have been pruned from a tree using drop.tip

Usage

```
CorrectRootTime(original.tree, pruned.tree)
```

Arguments

original.tree	A tree in phylo format.
pruned.tree	A tree in phylo format that represents a pruned version of original.tree.

Details

(NB: This function is designed to only cope with trees containing at least three tips.)

When removing taxa from a time-scaled tree using drop.tip in ape `$root.time` is left unchanged. This can cause downstream problems if not corrected and that is what this function does.

Note that `fixRootTime` in the `paleotree` package performs the same function, but is not called here to reduce the number of libraries on which `Claddis` is dependent. Interested users should also refer to the `dropPaleoTip` function in `paleotree`.

Value

Returns a tree (phylo object) with a corrected `$root.time`.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a simple four-taxon tree with branch lengths:
tree <- read.tree(text = "(A:1,(B:1,(C:1,D:1):1):1);")

# Set root age as 20 Ma:
tree$root.time <- 20

# Now prune taxon A:
pruned.tree <- drop.tip(tree, "A")

# Show that drop.tip has not updated the tree's root time:
pruned.tree$root.time

# Use the function to correct the root time:
pruned.tree <- CorrectRootTime(tree, pruned.tree)

# Show that the root time is now correct (19 Ma):
pruned.tree$root.time
```

Day2016 *Character-taxon matrix from Day et al. 2016*

Description

The character-taxon matrix from Day et al. (2016).

Format

A character-taxon matrix in the format imported by ReadMorphNexus.

References

Day, M. O., Rubidge, B. S. and Abdala, F., 2016. A new mid-Permian burnetiamorph therapsid from the Main Karoo Basin of South Africa and a phylogenetic review of Burnetiamorpha. Acta Palaeontologica Polonica, 61, 701-719.

`DiscreteCharacterRate` *Discrete character rates across trees, time, and character types*

Description

Given a tree and a cladistic-type matrix uses likelihood ratio tests to compare N-rate and 1-rate models across branches, clades, time bins, or character partitions.

Usage

```
DiscreteCharacterRate(tree, CladisticMatrix, TimeBins,
  BranchPartitionsToTest = NULL, CharacterPartitionsToTest = NULL,
  CladePartitionsToTest = NULL, TimeBinPartitionsToTest = NULL,
  ChangeTimes = "random", Alpha = 0.01,
  MultipleComparisonCorrection = "BenjaminiHochberg",
  PolymorphismState = "missing", UncertaintyState = "missing",
  InapplicableState = "missing", TimeBinApproach = "Lloyd",
  EnsureAllWeightsAreIntegers = FALSE, EstimateAllNodes = FALSE,
  EstimateTipValues = FALSE, InapplicablesAsMissing = FALSE,
  PolymorphismBehaviour = "equalp", UncertaintyBehaviour = "equalp",
  Threshold = 0.01)
```

Arguments

`tree`	A tree (phylo object) with branch lengths that represents the relationships of the taxa in `CladisticMatrix`.
`CladisticMatrix`	A character-taxon matrix in the format imported by ReadMorphNexus.
`TimeBins`	A vector of ages (in millions of years) indicating the boundaries of a series of time bins in order from oldest to youngest.
`BranchPartitionsToTest`	A list of branch(es) (edge numbers) to test for a 2-rate parameter model (i.e., one rate for the edge and another for the rest of the tree). If NULL (the default) then no partition test(s) will be made.
`CharacterPartitionsToTest`	A list of character partition(s) (character numbers) to test for a 2-rate parameter model (i.e., one rate for the partition and another for the remaining characters). If NULL (the default) then no partition test(s) will be made.
`CladePartitionsToTest`	A list of clade partition(s) (node numbers) to test for a 2-rate parameter model (i.e., one rate for the clade and another for the rest of the tree). If NULL (the default) then no partition test(s) will be made.

`TimeBinPartitionsToTest`
: A list of time bin partition(s) (numbered 1 to N) to test for a 2-rate parameter model (i.e., one rate for the time bin(s) and another for the remaining time bins). If NULL (the default) then no partition test(s) will be made.

`ChangeTimes`
: The time at which to record the character changes. One of `"midpoint"` (changes occur at the midpoint of the branch), `"spaced"` (changes equally spaced along branch), or `"random"` (change times drawn at random from a uniform distribution; the default and recommended option). Note: this is only meaningful if testing for time bin partitions.

`Alpha`
: The alpha value to be used for the significance tests. The default is 0.01.

`MultipleComparisonCorrection`
: One of `"BenjaminiHochberg"` (the Benjamini and Hochberg 1995 false discovery rate approach; default and recommended) or `"Bonferroni"` (the Bonferroni correction).

`PolymorphismState`
: One of `"missing"` (converts polymorphic values to NA; the default) or `"random"` (picks one of the possible polymorphic states at random).

`UncertaintyState`
: One of `"missing"` (converts uncertain values to NA; the default) or `"random"` (picks one of the possible uncertain states at random).

`InapplicableState`
: The only current option is `"missing"` (converts value to NA).

`TimeBinApproach`
: One of `"Close"` or `"Lloyd"` (the default).

`EnsureAllWeightsAreIntegers`
: Logical for whether (`TRUE`) to reweight non-integer weights until all weights are integers or to leave them as they are (`FALSE`; the default).

`EstimateAllNodes`
: Option passed to internal use of AncStateEstMatrix.

`EstimateTipValues`
: Option passed to internal use of AncStateEstMatrix.

`InapplicablesAsMissing`
: Option passed to internal use of AncStateEstMatrix.

`PolymorphismBehaviour`
: Option passed to internal use of AncStateEstMatrix.

`UncertaintyBehaviour`
: Option passed to internal use of AncStateEstMatrix.

`Threshold`
: Option passed to internal use of AncStateEstMatrix.

Details

Introduction

Morphological change can be captured by discrete characters and their evolution modelled as occurring along the branches of a phylogenetic tree. This function takes as primary input a character-taxon matrix of discrete characters (in the format imported by ReadMorphNexus) and a time-scaled

phylogenetic tree (in the format of **paleotree** or **strap**) and begins by inferring ancestral states at the tree's internal nodes using the AncStateEstMatrix function. From here changes along individual branches can be estimated (only the minimum number of changes are inferred; see GetAllStateChanges for an alternative but unfinished approach) and hence rates can be calculated.

A discrete character rate can be expressed as the mean number of changes per million years (users may wish to normalise this by the number of characters for interpretation) and can be calculated for a branch (edge) of the tree, a clade (a mean rate for the edges descended from a single node), a character partition (the mean rate for a subset of the characters across all edges), or, most complex (see Lloyd 2016), the mean rate across the edges (or parts of edges) present in a time bin (defined by two values denoting the beginning and end of the time bin). In an ideal scenario these rates could be compared at face value, but that would require a large number of characters and very minimal (or zero) missing data. I.e., at an extreme of missing data if only one character can be observed along a branch it will either change (the maximum possible rate of evolution) or it will not (the minimum possible rate of evolution). In such cases it would be unwise to consider either outcome as being a significant departure from the mean rate.

Because of these complications Lloyd et al. (2012) devised tests by which the significance of an edge (or other paritioning of the data, i.e., a clade, time bin etc.) could be considered to be significantly high or low in comparison to the mean rate for the whole tree (i.e., whether a two-rate model could be considered more likely than a one-rate model). This is achieved through a likelihood ratio test:

$$LR = valueoflikehoodfunctionunderthenull(one-rate)hypothesis/maximumpossiblevalueoflikehoodfunctionu$$

Typically we might expect the two hypotheses to be well defined a priori. E.g., an expectation that a specific branch of the tree might have a higher or lower rate than background due to some evolutionary shift. However, Lloyd et al. (2012) instead provided an exploratory approach whereby every possible one edge value was compared with the rate for the rest of the tree (and the equivalent with clades and time bins). This was the default in Claddis up to version 0.2, but this has now been replaced (since version 0.3) with a more customisable set of options that allows different types of hypotheses (e.g., partitioning the data by character), as well as more complex hypotheses (e.g., a three-rate model), to be tested.

The four types of rate hypothesis

Following Cloutier (1991), Lloyd (2016) extended the two main types of rate hypotheses to four:

1. A branch rate (available here with the `BranchPartitionsToTest` option).
2. A clade rate (available here with the `CladePartitionsToTest` option).
3. A time bin rate (available here with the `TimeBinPartitionsToTest` option).
4. A character partition rate (available here with the `CharacterPartitionsToTest` option).

In Claddis (>=0.3) these partitions are defined as a list of lists of vectors where only the first N - 1 partitions need be defined. E.g., if comparing the first edge value to the rest of the tree then the user only needs to define the value "1" and the function will automatically add a second partition containing all other edges. This can be set with the option `BranchPartitionsToTest = list(list(1))`. Similarly, to do what Lloyd et al. (2012) did and repeat the test for every edge in the tree (and assuming this variable is already named "tree") you could use, `BranchPartitionsToTest = lapply(as.list(1:nrow(tree$edge`

Because of the flexibility of this function the user can define any set of edges. For example, they could test whether terminal branches have a different rate from internal branches with `BranchPartitionsToTest = list(li`

The `CladePartitionsToTest` is really just a special subset of this type of hypothesis, but with edges being defined as descending from a specific internal node in the tree. Once again, an exploratory approach like that of Lloyd et al. (2012) can be used with: `CladePartitionsToTest = lapply(as.list(Ntip(tr`
Note that this excludes the root node as this would define a single partition and hence would represent the null hypothesis (a single rate model for the whole tree). More generally clades must be defined by the node numbers they correspond to. In R an easy way to identify these is with: `plot(tree); nodelabels()`.

Time bin partitions are defined in a similar way, but are numbered 1:N starting from the oldest time bin. So if wanting to do an exploratory test of single bin partitions (and only four time bins were specified) you could use: `TimeBinPartitionsToTest = lapply(as.list(1:4), as.list)`. Bins can be combined too, just as edges are above. For example, time bins 1 and 2 could form a single partition with: `TimeBinPartitionsToTest = list(list(1:2))`. Or if looking to test a model where each bin has its' own rate value you could use: `TimeBinPartitionsToTest = list(as.list(1:3))`. Note, as before we do not need to specify the fourth bin as this will be automatically done by the function, however, `TimeBinPartitionsToTest = list(as.list(1:4))` will also work. Some caution needs to be applied with N-rate models (where N is three or larger) as a result favouring such models does not necessarily endorse N-separate rates. I.e., it could simply be that one bin has such a large excursion that overall the N-rate model fits better than the 1-rate model, but some 2-rate models might be better still. It is up to the user to check this themselves by exploring smaller combinations of bins.

Finally, character partitions allow the user to explore whether rates vary across different character types, e.g., skeletal characters versus soft tissue characters, or cranial characters versus postcranial characters. Here characters are simply numbered 1:N, but here single character partitions are less likely to be of interest. As an example of use lets say the first ten characters are what we are interested in as a partition (the second partition being the remaining characters), we could use: `CharacterPartitionsToTest = list(list(1:10))` to test for a two-rate model.

Note that the list of lists structure is critical to defining partitions as it allows them to be of different sizes and numbers. For example, one partition of three and another of six, or one set of two partitions and another set of four partitions - structures not possible using vectors or matrices. However, it may not be intuitive to some users so it is recommended that the user refers to the examples above as a guide.

Additionally, it should be noted that the user can test multiple types of hypotheses simultaneously with the function. For example, performing several branch tests whilst also performing clade tests. However, it is not necessary to perform all types simultaneously (as was the case in version 0.2) and unused partition types can be set to NULL, the default in each case.

Other options

Since Claddis version 0.3 this function has allowed the user greater control with many more options than were offered previously and these should be considered carefully before running any tests.

Firstly, the user can pick an option for `ChangeTimes` which sets the times character changes are inferred to occur. This is only relevant when the user is performing time bin partition test(s) as this requires some inference to be made about when changes occur on branches that may span multiple time bins. The current options are: `"midpoint"` (all changes are inferred to occur midway along the branch, effectively mimicking the approach of Ruta et al. 2006), `"spaced"` (all changes are inferred to occur equally spaced along the branch, with changes occurring in character number order), or `"random"` (changes are assigned a random time by drawing from a uniform distribution between the beginning and end of each branch). The first of these is likely to lead to unrealistically "clumped"

changes and by extension implies completely correlated character change that would violate the assumptions of the Poisson distribution that underlies the significance tests here (Lloyd et al. 2012). At the other extreme, the equally spaced option will likely unrealistically smooth out time series and potnentially make it harder to reject the single-rate null. For these reasons, the random option is recommended and is set as the default. However, because it is random this makes the function stochastic (the answer can vary each time it is run) and so the user should therefore run the function multiple times if using this option (i.e., by using a for loop) and aggregating the results at the end (e.g., as was done by previous authors; Lloyd et al. 2012; Close et al. 2015).

Secondly, the `Alpha` value sets the significance threshold by which the likelihood ratio test's resulting p-value is compared. Following Lloyd et al. (2012) this is set lower (0.01) than the standard 0.05 value by default as those authors found rates to be highly heterogenous in their data set (fossil lungfish). However, this should not be adopted as a "standard" value without question. Note that the function also corrects for multiple comparisons (using the `MultipleComparisonCorrection` option) to avoid Type I errors (false positives). It does so (following Lloyd et al. 2012) using the Benjamini-Hochberg (Benjamini and Hochberg 1995) False Discovery Rate approach (see Lloyd et al. 2012 for a discussion of why), but the Bonferroni correction is also offered.

Thirdly, polymorphisms and uncertainities create complications for assessing character changes along branches. These can occur at the tips (true polymorphisms or uncertainties in sampled taxa) and internal nodes (uncertainty over the estimated ancestral state). There are two options presented here, and applicable to both `PolymorphismState` and `UncertaintyState` (allowing these to be set separately). These are to convert such values to missing (NA) or to pick one of the possible states at random. Using missing values will increase overall uncertainty and potentially lead to Type II errors (false negatives), but represents a conservative solution. The random option is an attempt to avoid Type II errors, but can be considered unrealistic, especially if there are true polymorphisms. Additionally, the random option will again make the function stochastic meaning the user should run it multiple times amd aggregate the results. Note that if there are no polymorphisms or uncertainties in the character-taxon matrix the latter can still occur with ancestral state estimates, espcially if the threshold value is set at a high value (see AncStateEstMatrix for details).

Fourthly, inapplicable characters can additionally complicate matters as they are not quite the same as missing data. I.e., they can mean that change in a particular character is not even possible along a branch. However, there is no easy way to deal with such characters at present so the user is not presented with a true option here - currently all inapplicable states are simply converted to missing values by the function. In future, though, other options may be available here. For now it is simply noted that users should be careful in making inferences if there are inapplicable characters in their data and should perhaps consider removing them with MatrixPruner to gauge their effect.

Fifthly, there are currenty two further options for assessing rates across time bins. As noted above a complication here is that character changes (the rate numerator) and character completeness (part of the rate denominator) are typically assessed on branches. However, branches will typically span time bin boundaries and hence many bins will contain only some portion of particular branches. The exact portion can be easily calculated for branch durations (the other part of the rate denominator) and the `ChangeTimes` option above is used to set the rate numerator, however, the completeness remains complex to deal with. The first attempt to deal with this was made by Close et al. (2015) who simply did weighted mean completeness by using the proportion of a branch in each bin as the weight and multiplying this by each branches completeness (the `"Close"` option here). However, this may lead to unrealistic "smoothing" of the data and perhaps more importantly makes no account of which characters are known in a bin. Lloyd (2016) proposed an alternative "subtree" approach which assesses completeness by considering each character to be represented by a subtree where

only branches that are complete are retained then branch durations in each bin are summed across subtrees such that the duration term automatically includes completenes (the "Lloyd" option here). Here the latter is strongly recommended, for example, because this will lead to the same global rate across the whole tree as the branch, clade or character partitions, whereas the Close approach will not.

Sixthly, all character changes are weighted according to the weights provided in the input character-taxon matrix. In many cases these will simply all be one, although see the equalise weights option in ReadMorphNexus. However, when weights vary they can create some issues for the function. Specifcally, changes are expected to be in the same (integer) units, but if weights vary then they have to be modelled accordingly. I.e., a character twice the weight of another may lead to a single change being counted as two changes. This is most problematic when the user has continuous characters which are automatically converted to gap-weighted (Thiele 1993) characters. However, this conversion creates drastically down-weighted characters and hence the user may wish to set the EnsureAllWeightsAreIntegers option to TRUE. Note that reweighting will affect the results and hence shifting the weights of characters up or down will necessarily lead to shifts in the relative Type I and II errors. This is an unexplored aspect of such approaches, but is something the user should be aware of. More broadly it is recommended that continuous (or gap-weighted) characters be avoided when using these approaches.

Finally, the ramaining options (EstimateAllNodes, EstimateTipValues, InapplicablesAsMissing, PolymorphismBehaviour, UncertaintyBehaviour, and Threshold) are all simply passed directly to AncStateEstMatrix for estimating the ancestral states and users should consult the help file for that function for further details.

Note that currently the function cannot deal with step matrices and that the terminal versus internal option from Brusatte et al. (2014) is yet to be implemented.

Value

`InferredCharacterChanges`
: Matrix of inferred character changes.

`IntrinsicCharacterRate`
: The intrinsic (global) character rate in changes per million years.

`ContinuousCharactersConvertedToDiscrete`
: Whether or not continuous characters were converted to discrete characters (important for handling the data in downstream analys(es)).

`BranchPartitionResults`
: List of branch partition results (corresponding to BranchPartitionsToTest. NULL if not requested.

`CharacterPartitionResults`
: List of character partition results (corresponding to CharacterPartitionsToTest. NULL if not requested.

`CladePartitionResults`
: List of clade partition results (corresponding to CladePartitionsToTest. NULL if not requested.

`TimeBinResults`
: List of time bin partition results (corresponding to TimeBinPartitionsToTest. NULL if not requested.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com> and Steve C. Wang <scwang@swarthmore.edu>

References

Benjamini, Y. and Hochberg, Y., 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal Statistical Society, Series B, 57, 289-300.

Brusatte, S. L., Lloyd, G. T., Wang, S. C. and Norell, M. A., 2014. Gradual assembly of avian body plan culminated in rapid rates of evolution across dinosaur-bird transition. Current Biology, 24, 2386-2392.

Close, R. A., Friedman, M., Lloyd, G. T. and Benson, R. B. J., 2015. Evidence for a mid-Jurassic adaptive radiation in mammals. Current Biology, 25, 2137-2142.

Cloutier, R., 1991. Patterns, trends, and rates of evolution within the Actinistia. Environmental Biology of Fishes, 32, 23–58.

Lloyd, G. T., 2016. Estimating morphological diversity and tempo with discrete character-taxon matrices: implementation, challenges, progress, and future directions. Biological Journal of the Linnean Society, 118, 131-151.

Lloyd, G. T., Wang, S. C. and Brusatte, S. L., 2012. Identifying heterogeneity in rates of morphological evolution: discrete character change in the evolution of lungfish (Sarcopterygii; Dipnoi). Evolution, 66, 330-348.

Ruta, M., Wagner, P. J. and Coates, M. I., 2006. Evolutionary patterns in early tetrapods. I. Rapid initial diversification followed by decrease in rates of character change. Proceedinsg of the Royal Society of London B, 273, 2107–2111.

Thiele, K.. 1993. The Holy Grail of the perfect character: the cladistic treatment of morphometric data. Cladistics, 9, 275-304.

Examples

```
# Set random seed:
set.seed(17)

# Generate a random tree for the Michaux data set:
tree <- rtree(nrow(Michaux1989$Matrix_1$Matrix))

# Update taxon names to match those in the data matrix:
tree$tip.label <- rownames(Michaux1989$Matrix_1$Matrix)

# Set root time by making youngest taxon extant:
tree$root.time <- max(diag(vcv(tree)))

# Get discrete character rates:
x <- DiscreteCharacterRate(tree = tree, CladisticMatrix =
  Michaux1989, TimeBins = seq(from = tree$root.time,
  to = 0, length.out = 5), BranchPartitionsToTest =
  lapply(as.list(1:nrow(tree$edge)), as.list),
  CharacterPartitionsToTest = lapply(as.list(1:3),
  as.list), CladePartitionsToTest =
```

```
lapply(as.list(Ntip(tree) + (2:Nnode(tree))),
as.list), TimeBinPartitionsToTest =
lapply(as.list(1:4), as.list), ChangeTimes =
"random", Alpha = 0.01, PolymorphismState =
"missing", UncertaintyState = "missing",
InapplicableState = "missing", TimeBinApproach =
"Lloyd")
```

`DolloSCM` *Stochastic Character Map For Dollo Character*

Description

Given a tree with binary tip states produces a stochastic Dollo character map.

Usage

```
DolloSCM(tree, tip.states)
```

Arguments

`tree`	A tree in phylo format with barnch lengths and a value for `$root.time`.
`tip.states`	A vector of tip states (must be 0 or 1) with names matching `tree$tip.label`.

Details

A non-ideal solution to the problem of generating a stochastic character map for a Dollo character (i.e., a single gain of the derived state (1) with any number of losses).

The function operates as follows:

1) Establishes least inclusive clade exhibiting the derived state (1). 2) Assumes single gain occurred on branch subtending this clade and with equal probability of occurring at any point along the branch. 3) Treats inclusive clade as a subtree and places a strong prior on the root of the derived state (1). 4) Calls `make.simmap` from the `phytools` package to generate a stochastic character map using a model where only losses are possible. 5) Outputs both the stochastic character map (time spent in each state on each branch) and a matrix of state changes.

(NB: As the map is stochastic the answer will be different each time the function is run and multiple replicates are strongly advised in order to ascertain uncertainty.)

This was the method used in Tarver et al. (2018).

Value

`Changes`	A matrix of all changes (gains and losses).
`SCM`	The stochastic character map.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Tarver, J. E., Taylor, R. S., Puttick, M. N., Lloyd, G. T., Pett, W., Fromm, B., Schirrmeister, B. E., Pisani, D., Peterson, K. J. and Donoghue, P. C. J., 2018. Well-annotated microRNAomes do not evidence pervasive miRNA loss. Genome Biology and Evolution, 6, 1457-1470.

Examples

```
# Create a random 10-taxon tree:
tree <- rtree(10)

# Arbitrarily add a root.time value of 100 Ma:
tree$root.time <- 100

# Generate random tip states (0s and 1s):
tip.states <- sample(c(0, 1), 10, replace = TRUE)

# Add labels to tip states:
names(tip.states) <- tree$tip.label

# Get a single stochastic character map:
out <- DolloSCM(tree, tip.states)

# View matrix of changes:
out$Changes

# View stochastic character map (time spent in each state on each branch):
out$SCM
```

EdgeLengthsInBins *Edge-lengths present in time-bins*

Description

Given a time-scaled tree and set of time bin boundaries will sum the edge-lengths present in each bin.

Usage

```
EdgeLengthsInBins(tree, time.bins, pruned.tree = NULL)
```

Arguments

tree	A time-scaled tree in phylo format with a $root.time value.
time.bins	A vector of ages in millions of years of time bin boundaries in old-to-young order.
pruned.tree	A time-scaled tree in phylo format with a $root.time value that is a subset of tree.

Details

Calculates the total edge length present in each of a series of time bins. This is intended as an internal function for rate calculations, but may be of use to someone.

The option of using a pruned.tree allows the user to correctly classify internal and terminal branches in a subtree of the larger tree. So for example, if taxa A and B are sisters then after pruning B the subtree branch leading to A is composed of an internal and a terminal branch on the complete tree.

Value

edge.length.in.bin
: A vector giving the summed values in millions of years for each time bin. Names indicate the maximum and minimum values for each time bin.

terminal.edge.length.in.bin
: As above, but counting terminal edges only.

internal.edge.length.in.bin
: As above, but counting internal edges only.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random 10-taxon tree:
tree <- rtree(10)

# Add root age:
tree$root.time <- 100

# Create time bins:
time.bins <- seq(100, 0, length.out = 11)

# Get edge lengths for each bin:
EdgeLengthsInBins(tree, time.bins)
```

EdgeMatch *Edge matching function*

Description

Given two trees where one is a pruned version of the other gives matching edges and nodes of pruned tree to original tree.

Usage

```
EdgeMatch(original.tree, pruned.tree)
```

Arguments

original.tree	A tree in phylo format.
pruned.tree	A tree in phylo format that represents a pruned version of original.tree.

Details

Finds matching edge(s) and node(s) for a pruned tree in the original tree from which it was created. This is intended as an internal function, but may be of use to someone.

Value

matching.edges	A list of the matching edges.
matching.nodes	A matrix of matching node numbers.
removed.edges	A vector of the removed edges.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random 10-taxon tree:
original.tree <- rtree(10)

# Remove three leaves:
pruned.tree <- drop.tip(original.tree, c("t1", "t3", "t8"))

# Find matching edges:
X <- EdgeMatch(original.tree, pruned.tree)

# Show matching edges:
X$matching.edges

# Show removed edges:
```

```
X$removed.edges
```

`FindAncestor` *Find ancestor*

Description

Finds the last common ancestor (node) of a set of two or more descendant tips.

Usage

```
FindAncestor(descs, tree)
```

Arguments

`descs`	A vector of mode character representing the tip names for which an ancestor is sought.
`tree`	The tree as a phylo object.

Details

Intended for use as an internal function for TrimMorphDistMatrix, but potentially of more general use.

Value

`anc.node`	The ancestral node number.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a simple four-taxon tree:
tree <- read.tree(text = "(A,(B,(C,D)));")

# Plot the tree:
plot(tree)

# Add nodelabels and show that the most recent common
# ancestor of B, C, and D is node 6:
nodelabels()

# Use FindAncestor to show that the most recent common
# ancestor of B, C, and D is node 6:
FindAncestor(c("B", "C", "D"), tree)
```

`FindLinkedEdges` *Find linked edges for a tree*

Description

Given a tree finds edges that are linked to each other.

Usage

```
FindLinkedEdges(tree)
```

Arguments

`tree`	A tree (phylo object).

Details

Finds all edges that link (share a node) with each edge of a tree.

This is intended as an internal function, but may be of use to someone else.

Value

Returns a matrix where links are scored 1 and everything else 0. The diagonal is left as zero.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a simple four-taxon tree:
tree <- read.tree(text="(A,(B,(C,D)));")

# Find linked (1) edges matrix for tree:
FindLinkedEdges(tree)
```

Gauthier1986 *Character-taxon matrix from Gauthier 1986*

Description

The character-taxon matrix from Gauthier (1986).

Format

A character-taxon matrix in the format imported by ReadMorphNexus.

References

Gauthier, J. A., 1986. Saurischian monophyly and the origin of birds. In Padian, K. (ed.) The Origin of Birds and the Evolution of Flight. Towne and Bacon, San Francisco, CA, United States, 1-55.

GetAllStateChanges *Finds all state changes on a tree using stochastic character mapping*

Description

Takes a cladistic matrix and time-scaled tree and makes point estimates for every character change using stochastic character mapping.

Usage

```
GetAllStateChanges(CladisticMatrix, Tree, TimeBins, NSimulations = 10,
  PolymorphismBehaviour = "equalp", UncertaintyBehaviour = "equalp",
  InapplicableBehaviour = "missing")
```

Arguments

CladisticMatrix	A character-taxon matrix in the format imported by ReadMorphNexus.
Tree	A time-scaled tree (phylo object) that represents the relationships of the taxa in CladisticMatrix.
TimeBins	A vector of ages representing the boundaries of a series of time bins.
NSimulations	The number of simulations to perform (passed to make.simmap.
PolymorphismBehaviour	What to do with polymorphic (&) characters. One of "equalp", "missing", or "random". See details.
UncertaintyBehaviour	What to do with uncertain (/) characters. One of "equalp", "missing", or "random". See details.

`InapplicableBehaviour`
: What to do with inapplicable characters. Only one option currently ("missing"). See details.

Details

Important: this function is not yet complete and should not be used.

A wrapper function for make.simmap in the phytools package.

This function is intended to enumerate all possible changes on a tree (including to and from missing or inapplicable states) under the assumptions of stochastic character mapping as an alternative means of establishing branch-lengths (for rate analyses) or recording the state occupied at a particular point in time for disparity analyses.

Value

`RootStates`
: A matrix of the root states for each character (column) and simulation (rows).

`AllStateChanges`
: A matrix of rows for each change with columns corresponding to the character, the simulation number, the edge number, the time the change occurred, and the start and end states.

`CharacterTimes`
: A vector of the sampled tree-length (in Ma) for each character.

`EdgeLengthsPerBin`
: A matrix of time bins (columns) and characters (rows) indicating the sampled tree-length (in Ma).

`TerminalEdgeLengthsPerBin`
: As above, but for terminal edges only.

`InternalEdgeLengthsPerBin`
: As above, but for internal edges only.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Set random seed:
set.seed(2)

# Use Day 2016 as source matrix:
CladisticMatrix <- Day2016

# Prune out continuous characters:
CladisticMatrix <- MatrixPruner(CladisticMatrix =
  CladisticMatrix, blocks2prune = 1)

# Prune out majority of characters so
# example runs quickly:
CladisticMatrix <- MatrixPruner(CladisticMatrix =
```

```
  CladisticMatrix, characters2prune = 1:32)

# Generete random tree for matrix taxa:
Tree <- rtree(nrow(Day2016$Matrix_1$Matrix))

# Add taxon names to tree:
Tree$tip.label <- rownames(Day2016$Matrix_1$Matrix)

# Add root age to tree:
Tree$root.time <- max(diag(vcv(Tree)))

# Get all state changes for two simulations:
StateChanges <-
  GetAllStateChanges(CladisticMatrix = CladisticMatrix,
  Tree = Tree, TimeBins = seq(Tree$root.time, 0,
  length.out = 3), NSimulations = 2)

# View matrix of all stochstic character changes:
StateChanges$AllStateChanges

# View vector of sampled time for each
# character:
StateChanges$CharacterTimes

# View matrix of edge lengths in each time bin:
StateChanges$EdgeLengthsPerBin

# View matrix of termnial edge lengths in each time bin:
StateChanges$TerminalEdgeLengthsPerBin

# View matrix of internal edge lengths in each time bin:
StateChanges$InternalEdgeLengthsPerBin
```

`GetDescendantEdges` *Gets descendant edges of an internal node*

Description

Returns all descendant edges of an internal node for a phylo object.

Usage

```
GetDescendantEdges(n, tree)
```

Arguments

`n`	An integer corresponding to the internal node for which the descendant edges are sought.
`tree`	A tree as a phylo object.

Details

Returns a vector of integers corresponding to row numbers in `$edge` or cells in `$edge.length` of the descendant edges of the internal node supplied.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create simple four-taxon tree:
tree <- read.tree(text = "(A,(B,(C,D)));")

# Plot tree:
plot(tree)

# Add nodelabels:
nodelabels()

# Add edgelabels (note that edges 5 and 6
# are descendants of node 7):
edgelabels()

# Use GdtDescendantEdges to show that edges
# 5 and 6 are descendants of node 7:
GetDescendantEdges(7, tree)
```

GetNodeAges *Returns node ages for a time-scaled tree*

Description

Given a tree with branch-lengths scaled to time and a value for `$root.time` will return a vector of node ages.

Usage

```
GetNodeAges(tree)
```

Arguments

`tree`	A tree (phylo object) with branch lengths representing time and a value for `$root.time`.

Details

Returns a vector of node ages (terminal and internal) labelled by their node number.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create simple four-taxon tree with edge lengths all
# set to 1 Ma:
tree <- read.tree(text="(A:1,(B:1,(C:1,D:1):1):1);")

# Set root.time as 10 Ma:
tree$root.time <- 10

# Get node ages:
GetNodeAges(tree)
```

MakeMorphMatrix *Creates a morphological data file from a matrix*

Description

Creates a morphological data file from a character-taxon matrix.

Usage

```
MakeMorphMatrix(CharacterTaxonMatrix, header = "", weights = NULL,
  ordering = NULL, symbols = NULL, equalise.weights = FALSE)
```

Arguments

CharacterTaxonMatrix	A Character-Taxon (columns-rows) matrix, with taxon names as rownames.
header	A scalar indicating any header text (defaults to an empty string: "").
weights	A vector specifying the weights used (if not specified defaults to 1).
ordering	A vector indicating whether characters are ordered ("ord") or unordered ("unord") (if no specified defaults to ordered).
symbols	The symbols to use if writing to a file (defaults to the numbers 0:9 then the letters A to V).
equalise.weights	Optional that overrides the weights specified above make all characters truly equally weighted.

Details

Claddis generally assumes that matrices will be imported into R from the #NEXUS format, but in some cases (e.g., when using simulated data) it might be desirable to build a matrix within R. This function allows the user to convert such a matrix into the format required by other Claddis functions as long as it only contains a single block.

NB: Currently the function cannot deal directly with step matrices or continuous characters.

Value

`Topper`	Contains any header text or step matrices and pertains to the entire file.
`Matrix_N`	One or more matrix blocks (numbered 1 to N) with associated information pertaining only to that matrix block. This includes the block name (if specificed, NA if not), the block datatype (one of "CONTINUOUS", "DNA", "NUCLEOTIDE", "PROTEIN", "RESTRICTION", "RNA", or "STANDARD"), the actual matrix (taxa as rows, names stored as rownames and characters as columns), the ordering type of each character ("ord" = ordered, "unord" = unordered), the character weights, the minimum and maximum values (used by Claddis' distance functions), and the original characters (symbols, missing, and gap values) used for writing out the data.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

ReadMorphNexus

Examples

```
# Create random 10-by-50 matrix:
CharacterTaxonMatrix <- matrix(sample(c("0", "1", "0&1", NA, ""),
  500, replace = TRUE), nrow = 10, dimnames =
  list(apply(matrix(sample(LETTERS, 40,
  replace = TRUE), nrow = 10), 1, paste,
  collapse = ""), c()))

# Reformat for use elsewhere in Claddis:
MakeMorphMatrix(CharacterTaxonMatrix)
```

`MatrixPruner` *Prunes a character matrix of characters or taxa*

Description

Prunes a character matrix of characters, taxa, or both.

Usage

```
MatrixPruner(CladisticMatrix, blocks2prune = c(),
  characters2prune = c(), taxa2prune = c(), removeinvariant = FALSE)
```

Arguments

`CladisticMatrix`	The cladistic matrix in the format imported by ReadMorphNexus.
`blocks2prune`	A vector of number(s) of any blocks to prune.
`characters2prune`	A vector of character numbers to prune.
`taxa2prune`	A vector of taxon names to prune (these must be present in `rownames(CladisticMatrix$matrix)`.
`removeinvariant`	A logical for whether invariant characters should (TRUE) or should not (FALSE, default) be pruned.

Details

Removing characters or taxa from a matrix imported using ReadMorphNexus is not simple due to associated vectors for ordering, character weights etc. To save repetitively pruning each part this function takes the matrix as input and vector(s) of either block numbers, character numbers, taxon names, or any combination thereof and returns a matrix with these items removed. Minimum and maximum values (used by MorphDistMatrix) are also updated and the user has the option to remove constant characters this way as well (e.g, to reduce the memory required for a DNA matrix).

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

ReadMorphNexus

Examples

```
# Remove the outgroup taxon and characters 11 and 53 from Gauthier1986:
prunedmatrix <- MatrixPruner(CladisticMatrix = Gauthier1986, characters2prune = c(11, 53),
  taxa2prune = c("Outgroup"))

# Show priuned matrix:
prunedmatrix$Matrix_1$Matrix
```

Michaux1989 *Character-taxon matrix from Michaux 1989*

Description

The character-taxon matrix from Michaux (1989).

Format

A character-taxon matrix in the format imported by ReadMorphNexus.

References

Michaux, B., 1989. Cladograms can reconstruct phylogenies: an example from the fossil record. Alcheringa, 13, 21-36.

MinSpanTreeEdges *Get edges of minimum spanning tree*

Description

Returns edges of a minimum spanning tree given a distance matrix.

Usage

```
MinSpanTreeEdges(dist.matrix)
```

Arguments

dist.matrix	A square matrix of distances between objects.

Details

This function is a wrapper for mst in the ape package, but returns a vector of edges rather than a square matrix of links.

Value

A vector of named edges (X->Y) with their distances. The sum of this vector is the length of the minimum spanning tree.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a simple square matrix of distances:
dist.matrix <- matrix(c(0,1,2,3,1,0,1,2,2,1,0,1,3,2,1,0), nrow = 4,
  dimnames = list(LETTERS[1:4], LETTERS[1:4]))

# Show matrix to confirm that the off diagonal has the shortest
# distances:
dist.matrix

# Use MinSpanTreeEdges to get the edges for the minimum spanning
# tree:
MinSpanTreeEdges(dist.matrix)

# Use sum of MinSpanTreeEdges to get the length of the minimum
# spanning tree:
sum(MinSpanTreeEdges(dist.matrix))
```

MorphDistMatrix *Get distance matrices from a cladistic matrix*

Description

Takes a cladistic morphological dataset and converts it into a set of pairwise distances.

Usage

```
MorphDistMatrix(CladisticMatrix, Distance = "MORD", GEDType = "Wills",
  TransformDistances = "arcsine_sqrt",
  PolymorphismBehaviour = "min.difference",
  UncertaintyBehaviour = "min.difference",
  InapplicableBehaviour = "missing", CharacterDependencies = NULL,
  Alpha = 0.5)
```

Arguments

`CladisticMatrix`
: A character-taxon matrix in the format imported by ReadMorphNexus.

`Distance`
: The distance metric to use. Must be one of `"GC"`, `"GED"`, `"RED"`, or `"MORD"` (the default).

`GEDType`
: The type of GED to use. Must be one of `"Legacy"`, `"Hybrid"`, or `"Wills"` (the default). See details for an explanation.

`TransformDistances`
: Whether to transform the distances. Options are `"none"`, `"sqrt"`, or `"arcsine_sqrt"` (the default). (Note: this is only really appropriate for the proportional distances, i.e., "GC" and "MORD".)

`PolymorphismBehaviour`
: The distance behaviour for dealing with polymorphisms. Must be one of `"mean.difference"`, `"min.difference"` (the default), or `"random"`.

`UncertaintyBehaviour`
: The distance behaviour for dealing with uncertainties. Must be one of `"mean.difference"`, `"min.difference"` (the default), or `"random"`.

`InapplicableBehaviour`
: The behaviour for dealing with inapplicables. Must be one of `"missing"` (default), or `"HSJ"` (Hopkins and St John 2018; see details).

`CharacterDependencies`
: Only relevant if using `InapplicableBehaviour = "HSJ"`. Must be a two-column matrix with colnames "DependentCharacter" and "IndependentCharacter" that specifies character hierarchies. See details.

`Alpha`
: The alpha value (sensu Hopkins and St John 2018). Only relevant if using `InapplicableBehaviour = "HSJ"`. See details.

Details

There are many options to consider when generating a distance matrix from morphological data, including the metric to use, how to treat inapplicable, polymorphic (e.g., 0&1), or uncertain (e.g., 0/1) states, and whether the output should be transformed (e.g., by taking the square root so that the distances are - or approximate - Euclidean distances). Some of these issues have been discussed previously in the literature (e.g., Lloyd 2016; Hopkins and St John 2018), but all likely require further study.

Claddis currently offers four different distance metrics: 1. Raw Euclidean Distance (`RED`) - this is only really applicable if there are no missing data, 2. The Gower Coefficient (`GC`; Gower 1971) - this rescales distances by the number of characters that can be coded for both taxa in each pairwise comparison thus correcting for missing data, 3. The Maximum Observable Rescaled Distance (`MORD`) - this was introduced by Lloyd (2016) as an extension of the `GC` designed to deal with the fact that multistate ordered characters can lead to `GC`s of greater than 1 and works by rescaling by the maximum possible distance that could be observed based on the number of characters codable in each pairwise comparison meaning all resulting distances are on a zero to one scale, and 4. The Generalised Euclidean Distance - this was introduced by Wills (1998) as a means of correcting for the fact that a `RED` metric will become increasingly non-Euclidean as the amount of missing data increases and works by filling in missing distances (for characters that are coded as missing in at least

one taxon in the pairwise comparison) by using the mean pairwise dissimilarity for that taxon pair as a substitute. In effect then, `RED` makes no consideration of missing data, `GC` and `MORD` normalise by the available data (and are identical if there are no ordered multistate characters), and `GED` fills in missing distances by extrapolating from the available data.

Note that Lloyd (2016) misidentified the substitute dissimilarity for the `GED` as the mean for the whole data set (Hopkins and St John 2018) and this was the way the GED implementation of Claddis operated up to version 0.2. This has now been amended (as of version 0.3) so that the function produces the `GED` in the form that Wills (1998) intended. However, this implementation can still be accessed as the `Legacy` option for `GEDType`, with `Wills` being the WIlls (1998) implementation. An advantage of this misinterpreted form of `GED` is that it will always return a complete pairwise distance matrix, however it is not recommended (see Lloyd 2016). Instead a third option for `GEDType` - (`Hybrid`) - offers the same outcome but only uses the mean distance from the entire matrix in the case where there are no codable characters in common in a pairwise comparison. This new hybrid option has not been used in a published study.

Typically the resulting distance matrix will be used in an ordination procedure such as principal coordinates (effectively classical multidimensional scaling where k, the number of axes, is maximised at N - 1, where N is the number of rows (i.e., taxa) in the matrix). As such the distance should be - or approximate - Euclidean and hence a square root transformation is typically applied (`TransformDistances` with the `sqrt` option). However, if applying pre-ordination (i.e., ordination-free) disparity metrics (e.g., weighted mean pairwise distance) you may wish to avoid any transformation (`none` option). In particular the `MORD` will only fall on a zero to one scale if this is the case. However, if transforming the `MORD` for ordination this zero to one property may mean the arcsine square root (`arcsine_sqrt` option) is preferred. (Note that if using only unordered multistate or binary characters and the `GC` the zero to one scale will apply too.)

An unexplored option in distance matrix construction is how to deal with polymorphisms (Lloyd 2016). Up to version 0.2 of Claddis all polymorphisms were treated the same regardless of whether they were true polymorphisms (multiple states are observed in the taxon) or uncertainties (multiple, but not all states, are posited for the taxon). Since version 0.3, however, these two forms can be distinguished by using the different #NEXUS forms (Maddison et al. 1997), i.e., (01) for polymorphisms and {01} for uncertainties and within Claddis these are represented as 0&1 or 0/1, respectively. Thus, since 0.3 Claddis allows these two forms to be treated separately, and hence differently (with `PolymorphismBehaviour` and `UncertaintyBehaviour`). Again, up to version 0.2 of Claddis no options for polymorphism behaviour were offered, instead only a minimum distance was employed. I.e., the distance between a taxon coded 0&1 and a taxon coded 2 would be the smaller of the comparisons 0 with 2 or 1 with 2. Since version 0.3 this is encoded in the `min.difference` option. Currentlly two alternatives (`mean.difference` and `random`) are offered. The first takes the mean of each possible difference and the second simply samples one of the states at random. Note this latter option makes the function stochastic and so it should be rerun multiple times (for example, with a `for` loop or `apply` function). In general this issue (and these options) are not explored in the literature and so no recommendation can be made beyond that users should think carefully about what this choice may mean for their individual data set(s) and question(s).

A final consideration is how to deal with inapplicable characters. Up to version 0.2 Claddis treated inapplicable and missing characters the same (as NA values, i.e., missing data). However, since Claddis version 0.3 these can be imported separately, i.e., by using the "MISSING" and "GAP" states in #NEXUS format (Maddison et al. 1997), with the latter typically representing the inapplicable character. These appear as NA and empty strings (""), respectively, in Claddis format. Hopkins and St John (2018) showed how inapplicable characters - typically assumed to represent

secondary characters - could be treated in generating distance matrices. These are usually hierarchical in form. E.g., a primary character might record the presence or absence of feathers and a secondary character whether those feathers are symmetric or asymmetric. The latter will generate inapplicable states for taxa without feathers and without correcting for this ranked distances can be incorrect (Hopkins and St John 2018). Unfortunately, however, the #NEXUS format (Maddison et al. 1997) does not really allow explicit linkage between primary and secondary characters and so this information must be provided separately to use the Hopkins and St John (2018) approach. This is done here with the `CharacterDependencies` option. This must be in the form of a two-column matrix with column headers of "DependentCharacter" and "IndependentCharacter". The former being secondary characters and the latter the corresponding primary character. (Note that characters are to be numbered across the whole matrix from 1 to N and do not restart with each block of the matrix.) If using `InapplicableBehaviour = "HSJ"` the user must also provide an `Alpha` value between zero and one. When `Alpha = 0` the secondary characters contribute nothing to the distance and when `Alpha = 1` the primary character is not counted in the weight separately (see Hopkins and St John 2018). The default value (0.5) offers a compromise bteween these two extremes.

Here the implementation of this approach differs somewhat from the code available in the supplementary materials to Hopkins and St John (2018). Specifically, this approach is incorporated (and used) regardless of the overriding distance metric (i.e., the `Distance` option). Additionally, the Hopkins and St John function specifically allows an extra level of dependency (secondary and tertary characters) with these being applied recursively (tertiary first then secondary). Here, though, additional levels of dependency do not need to be defined by the user as this information is already encoded in the `CharacterDependencies` option. Furthermore, because of this any level of dependency is possible (if unlikely).

Value

`DistanceMetric` The distance metric used.

`DistanceMatrix` The distance matrix returned.

`ComparableCharacterMatrix`
The matrix of characters that can be compared for each pairwise distance.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com> and Thomas Guillerme <guillert@tcd.ie>

References

Gower, J. C., 1971. A general coefficient of similarity and some of its properties. Biometrics 27, 857–871.

Hopkins, M. J. and St John, K., 2018. A new family of dissimilarity metrics for discrete character matrices that include inapplicable characters and its importance for disparity studies. Proceedings of the Royal Society of London B, 285, 20181784.

Lloyd, G. T., 2016. Estimating morphological diversity and tempo with discrete character-taxon matrices: implementation, challenges, progress, and future directions. Biological Journal of the Linnean Society, 118, 131-151.

Maddison, D. R., Swofford, D. L. and Maddison, W. P., 1997. NEXUS: an extensible file format for systematic information. Systematic Biology, 46, 590-621.

Wills, M. A., 1998. Crustacean disparity through the Phanerozoic: comparing morphological and stratigraphic data. Biological Journal of the Linnean Society, 65, 455-500.

Examples

```
# Get morphological distances for the Day et
# al. (2016) data set:
distances <- MorphDistMatrix(Day2016)

# Show distance metric:
distances$DistanceMetric

# Show distance matrix:
distances$DistanceMatrix

# Show number of characters that can be scored for
# each pairwise comparison:
distances$ComparableCharacterMatrix

# To repeat using the Hopkins and St John approach
# we first need to define the character dependency
# (here there is only one, character 8 is a
# secondary where 7 is the primary character):
CharacterDependencies <- matrix(c(8, 7), ncol = 2,
  byrow = TRUE, dimnames = list(c(),
  c("DependentCharacter",
  "IndependentCharacter")))

# Get morphological distances for the Day et
# al. (2016) data set using HSJ approach:
distances <- MorphDistMatrix(Day2016,
  InapplicableBehaviour = "HSJ",
  CharacterDependencies = CharacterDependencies,
  Alpha = 0.5)

# Show distance metric:
distances$DistanceMetric

# Show distance matrix:
distances$DistanceMatrix

# Show number of characters that can be scored for
# each pairwise comparison:
distances$ComparableCharacterMatrix
```

MorphMatrix2PCoA *Principal Coordinates on a Cladistic Matrix*

Description

Performs Principal Coordinates Analysis (PCoA) on a cladistic matrix.

Usage

```
MorphMatrix2PCoA(CladisticMatrix, Distance = "MORD", GEDType = "Wills",
  TransformDistances = "arcsine_sqrt",
  DistPolymorphismBehaviour = "min.difference",
  DistUncertaintyBehaviour = "min.difference",
  DistInapplicableBehaviour = "missing", CharacterDependencies = NULL,
  Alpha = 0.5, correction = "cailliez", Tree = NULL,
  EstimateAllNodes = FALSE, EstimateTipValues = FALSE,
  InapplicablesAsMissing = FALSE,
  AncestralPolymorphismBehaviour = "equalp",
  AncestralUncertaintyBehaviour = "equalp", Threshold = 0.01)
```

Arguments

`CladisticMatrix`
: A vector of mode character representing the tip names for which an ancestor is sought.

`Distance`
: The distance method to use (one of "RED", "GED", "GC", or "MORD" - the default). See MorphDistMatrix for more details.

`GEDType`
: The type of GED use. Must be one of `"Legacy"`, `"Hybrid"`, or `"Wills"` (the default). See details for an explanation.

`TransformDistances`
: The transformation to apply to distances. See MorphDistMatrix for details.

`DistPolymorphismBehaviour`
: The distance behaviour for dealing with polymorphisms. Must be one of `"mean.difference"`, `"min.difference"` (the default), or `"random"`. See MorphDistMatrix for details.

`DistUncertaintyBehaviour`
: The distance behaviour for dealing with uncertainties. Must be one of `"mean.difference"`, `"min.difference"` (the default), or `"random"`. See MorphDistMatrix for details.

`DistInapplicableBehaviour`
: The behaviour for dealing with inapplicables. Must be one of `"missing"` (default), or `"HSJ"`. See MorphDistMatrix for details.

`CharacterDependencies`
: Only relevant if using `InapplicableBehaviour = "HSJ"`. Must be a two-column matrix with colnames "DependentCharacter" and "IndependentCharacter" that specifies character hierarchies. See MorphDistMatrix for details.

`Alpha`
: The alpha value (sensu Hopkins and St John 2018). Only relevant if using `InapplicableBehaviour = "HSJ"`. See MorphDistMatrix for details.

`correction`
: The negative eigenvalue correction to use (one of "lingoes", "none", or "cailliez" - the default). See pcoa for more details.

`Tree` If a phylmorphospace is desired then a tree with root age and branch-lengths must be included.

`EstimateAllNodes`
If including a tree whether you want to estinate ancestral states for all characters (default is FALSE). See AncStateEstMatrix for more details.

`EstimateTipValues`
If including a tree whether you want to estinate missing or polymorphic tip states (default is FALSE). See AncStateEstMatrix for more details.

`InapplicablesAsMissing`
See AncStateEstMatrix.

`AncestralPolymorphismBehaviour`
Behaviour for dealing with polymorphisms when producing ancestral state estimates - see AncStateEstMatrix.

`AncestralUncertaintyBehaviour`
Behaviour for dealing with uncertainties when producing ancestral state estimates - see AncStateEstMatrix.

`Threshold` Threshold for ancestral state estimation of discrete characters - see AncStateEstMatrix for details.

Details

Takes a cladistic matrix in the format imported by ReadMorphNexus and performs Principal Coordinates (Gower 1966) analysis on it.

This function is effectively a wrapper for pcoa from the ape package and the user is referred there for some of the options (e.g., using the Caillez 1983 approach to avoiding negative eigenvalues).

If providing a tree and inferring ancestral states then options to also infer missing or uncertain tips and whether to infer values for all characters at all internal nodes are provided (via AncStateEstMatrix).

Other options within the function concern the distance metric to use and the transformation to be used if selecting a propotional distance (see MorphDistMatrix).

IMPORTANT: The function can remove taxa (or if including a tree, nodes as well) if they lead to an incomplete distance matrix (see TrimMorphDistMatrix).

Value

`Tree` The tree (if supplied). Note this may be pruned from the input tree by TrimMorphDistMatrix.

`DistMatrix` The distance matrix. Note this may be pruned by TrimMorphDistMatrix and thus not include all taxa.

`RemovedTaxa` A vector of taxa (or nodes) removed by TrimMorphDistMatrix. Returns NULL if none are removed.

`note` See pcoa.

`values` See pcoa.

`vectors` See pcoa.

`trace` See pcoa.

`vectors.cor` See pcoa.

`trace.cor` See pcoa.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Cailliez, F., 1983. The analytical solution of the additive constant problem. Psychometrika, 48, 305-308.

Gower, J. C., 1966. Some distance properties of latent root and vector methods used in multivariate analysis. Biometrika, 53, 325-338.

Examples

```
# Run on Michaux (189) data set with defaults:
x <- MorphMatrix2PCoA(Michaux1989)

# Show output:
x

# Generate a (made up) tree:
Tree <- rtree(length(rownames(Michaux1989$Matrix_1$Matrix)))

# Add taxon names to it:
Tree$tip.label <- rownames(Michaux1989$Matrix_1$Matrix)

# Set root time by making youngest taxon extant:
Tree$root.time <- max(diag(vcv(Tree)))

# Run with tree:
y <- MorphMatrix2PCoA(Michaux1989, Tree = Tree)

# Show new output:
y
```

MorphospacePlot *Plot Morphopace*

Description

Plots a morphospace using the output from MorphMatrix2PCoA.

Usage

```
MorphospacePlot(pcoa_input, x_axis = 1, y_axis = 2, z_axis = NULL,
  plot_taxon_names = FALSE, plot_internal_nodes = FALSE,
  plot_root = TRUE, root_colour = "grey")
```

Arguments

pcoa_input	The main input in the format output from MorphMatrix2PCoA.
x_axis	Which ordination axis to plot as the x-axis (defaults to 1).
y_axis	Which ordination axis to plot as the y-axis (defaults to 2).
z_axis	Which ordination axis to plot as the z-axis (defaults to NULL, i.e., is not plotted).
plot_taxon_names	Optional to plot the names of the taxa (defaults to FALSE).
plot_internal_nodes	Optional to plot the internal nodes of the tree (if included in pcoa_input) (defaults to FALSE).
plot_root	Optional to plot the root separately (defaults to FALSE).
root_colour	If plotting the root separately (previous option) sets the root colour.

Details

Uses output from MorphMatrix2PCoA as input.

Allows plotting of a third axis using the technique of Matthew Wills (Wills et al. 1994; their Figures 4 and 8; Wills 1998; his Figure 4), where black and white indicate positive and negative values respectovely, and the size of points there magnitudes.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com> and Emma Sherratt <emma.sherratt@gmail.com>

References

Wills, M. A., 1998. Cambrian and Recent disparity: the picture from priapulids. Paleobiology, 24, 177-199.

Wills, M. A., Briggs, D. E. G. and Fortey, R. A., 1994. Disparity as an evolutionary index: a comparison of Cambrian and Recent arthropods. Paleobiology, 20, 93-130.

Examples

```
# Set random seed:
set.seed(4)

# Generate a random tree for the Michaux 1989 data set:
tree <- rtree(length(rownames(Michaux1989$Matrix_1$Matrix)))

# Add taxon names to the tree:
tree$tip.label <- rownames(Michaux1989$Matrix_1$Matrix)

# Perform a phylogenetic Principal Coordinates Analysis:
pcoa_input <- MorphMatrix2PCoA(Michaux1989, Tree = tree)
```

```
# Plot the results:
MorphospacePlot(pcoa_input, plot_taxon_names = TRUE)
```

`MultiMorphospacePlot` *Plot Multiple Morphopaces*

Description

Plots morphospaces for any number of axes.

Usage

```
MultiMorphospacePlot(pcoa_input, N_axes = 4, plot_taxon_names = FALSE,
  plot_internal_nodes = FALSE, plot_root = TRUE,
  root_colour = "grey")
```

Arguments

`pcoa_input`	The main input in the format output from MorphMatrix2PCoA.
`N_axes`	An integer indicating the total number of axes to plot (should minimally be three).
`plot_taxon_names`	Optional to plot the names of the taxa (defaults to FALSE).
`plot_internal_nodes`	Optional to plot the internal nodes of the tree (if included in `pcoa_input`) (defaults to FALSE).
`plot_root`	Optional to plot the root separately (defaults to FALSE).
`root_colour`	If plotting the root separately (previous option) sets the root colour.

Details

Takes output from MorphMatrix2PCoA as input and serves as a wrapper function for MorphospacePlot.

Plots multiple bivariate plots of ordination axes starting with 1 and up to N, such that every possible bivariate plot is produced.

Author(s)

Emma Sherratt <emma.sherratt@gmail.com> and Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create PCOA data:
pcoa_input <- MorphMatrix2PCoA(Michaux1989)

# Plot first three axes:
MultiMorphospacePlot(pcoa_input, N_axes = 3)
```

`PhyloCharCompletenessInBins` *Phylogenetic character completeness in time-bins*

Description

Given a cladistic matrix, time-scaled tree, and set of time bin boundaries will return the proportional character completeness in each bin.

Usage

```
PhyloCharCompletenessInBins(CladisticMatrix, TimeTree, TimeBins,
  plot = FALSE, CI = 0.95)
```

Arguments

`CladisticMatrix`	A cladistic matrix in the form imported by ReadMorphNexus.
`TimeTree`	A time-scaled phylogenetic tree containing all the taxa in `CladisticMatrix`.
`TimeBins`	A set of time bin boundaries (oldest to youngest) in millions of years.
`plot`	An optional choice to plot the results (default is `FALSE`).
`CI`	The confidence interval to be used as a proportion (0 to 1). Default is 0.95 (i.e., 95%).

Details

Character completeness metrics have been used as an additional metric for comparing fossil record quality across time, space, and taxa. However, these only usually refer to point samples of fossils in bins, and not our ability to infer information along the branches of a phylogenetic tree.

This function returns the proportional phylogenetic character completeness for a set of time bins.

Value

A list summarising the mean, upper and lower 95

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random tree for the Day et al. 2016 data set:
Day2016tree <- rtree(nrow(Day2016$Matrix_1$Matrix))
Day2016tree$tip.label <- rownames(Day2016$Matrix_1$Matrix)
Day2016tree$root.time <- max(diag(vcv(Day2016tree)))

# Get proportional phylogenetic character completeness in ten equal-length
# time bins:
PhyloCharCompletenessInBins(CladisticMatrix = Day2016,
  TimeTree = Day2016tree, TimeBins = seq(from =
  Day2016tree$root.time, to = Day2016tree$root.time -
  max(diag(vcv(Day2016tree))), length.out = 11))
```

PlotCharacterChanges *Plots character changes on branches*

Description

Plots character changes in boxes on branches.

Usage

```
PlotCharacterChanges(character.changes, tree)
```

Arguments

`character.changes`	A matrix of character changes.
`tree`	Tree on which character changes occur.

Details

Takes the `character.changes` output from DiscreteCharacterRate and plots it on the tree used to generate it.

Value

A plot of character changes on a tree.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Set random seed:
set.seed(17)

# Generate a random tree for the Michaux data set:
tree <- rtree(nrow(Michaux1989$Matrix_1$Matrix))

# Update taxon names to match those in the data matrix:
tree$tip.label <- rownames(Michaux1989$Matrix_1$Matrix)

# Set root time by making youngest taxon extant:
tree$root.time <- max(diag(vcv(tree)))

# Get discrete character rates (includes changes):
out <- DiscreteCharacterRate(tree, Michaux1989,
  seq(tree$root.time, 0, length.out = 3),
  BranchPartitionsToTest = list(list(1)), Alpha = 0.01)

# Plot character changes on the tree:
PlotCharacterChanges(out$InferredCharacterChanges,
  tree)
```

ReadMorphNexus *Reads in a morphological #NEXUS data file*

Description

Reads in a morphological data file in #NEXUS format.

Usage

```
ReadMorphNexus(File, EqualiseWeights = FALSE)
```

Arguments

`File`	A file name specified by either a variable of mode character, or a double-quoted string.
`EqualiseWeights`	Optional that overrides the weights specified in the file to make all characters truly equally weighted.

Details

Reads in a #NEXUS (Maddison et al. 1997) data file representing the distribution of characters (continuous, discrete, DNA etc.) in a set of taxa. Unlike read.nexus.data this function can handle polymorphisms (e.g., `(012)`).

Note that the function is generally intolerant to excursions from a standard format and it is recommended your data be formatted like the `morphmatrix.nex` example below. However, the function also produces informative error messages if (expected) excursions are discovered.

Previously all empty values (missing or inapplicable) were treated as NAs. But now anything coded as a "gap" now appears as an empty text string ("") in the matrix. Additionally, previously polymorphisms and uncertianties were both considered as polymorphisms with multiple states separated by an ampersand ("&"), but now polymorphisms use the ampersand ("&") and uncertainties use a slash ("/"), allowing for different treatment later and correct outputting when writing to #NEXUS format. (NB: TNT does not allow this distinction and so both polymorphisms and uncertainties will be output as polymorphisms.)

Value

`Topper`	Contains any header text or step matrices and pertains to the entire file.
`Matrix_N`	One or more matrix blocks (numbered 1 to N) with associated information pertaining only to that matrix block. This includes the block name (if specificed, NA if not), the block datatype (one of "CONTINUOUS", "DNA", "NUCLEOTIDE", "PROTEIN", "RESTRICTION", "RNA", or "STANDARD"), the actual matrix (taxa as rows, names stored as rownames and characters as columns), the ordering type of each character ("ord" = ordered, "unord" = unordered), the character weights, the minimum and maximum values (used by Claddis' distance functions), and the original characters (symbols, missing, and gap values) used for writing out the data.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P., 1997. NEXUS: an extensible file format for systematic information. Systematic Biology, 46, 590-621.

See Also

read.nexus.data

Examples

```
# Create example matrix (NB: creates a file in the current
# working directory called "morphmatrix.nex"):
cat("#NEXUS\n\nBEGIN DATA;\n\tDIMENSIONS  NTAX=5 NCHAR=5;\n\t
FORMAT SYMBOLS= \" 0 1 2\" MISSING=? GAP=- ;\nMATRIX\n\n
Taxon_1  010?0\nTaxon_2  021?0\nTaxon_3  02111\nTaxon_4  011-1
\nTaxon_5  001-1\n;\nEND;\n\nBEGIN ASSUMPTIONS;\n\t
OPTIONS  DEFTYPE=unord PolyTcount=MINSTEPS ;\n\t
TYPESET * UNTITLED  = unord: 1 3-5, ord: 2;\n\t
WTSET * UNTITLED  = 1: 2, 2: 1 3-5;\nEND;", file = "morphmatrix.nex")
```

```
# Read in example matrix:
morph.matrix <- ReadMorphNexus("morphmatrix.nex")

# View example matrix in R:
morph.matrix

# Remove the generated data set:
file.remove("morphmatrix.nex")
```

SafeTaxonomicReduction

Safe Taxonomic Reduction

Description

Performs Safe Taxonomic Reduction (STR) on a character-taxon matrix.

Usage

```
SafeTaxonomicReduction(CladisticMatrix)
```

Arguments

CladisticMatrix	A character-taxon matrix in the format imported by ReadMorphNexus.

Details

Performs Safe Taxonomic Reduction (Wilkinson 1995).

If no taxa can be safely removed will print the text "No taxa can be safely removed", and the `str.list` and `removed.matrix` will have no rows.

NB: If your data contains inapplicable characters these will be treated as missing data, but this is inappropriate. Thus the user is advised to double check that any removed taxa make sense in the light of inapplicable states. (As far as I am aware this same behaviour occurs in the TAXEQ3 software.)

Value

str.list	A matrix listing the taxa that can be removed (Junior), the taxa which they are equivalent to (Senior) and the rule under which they can be safely removed (Rule).
reduced.matrix	A character-taxon matrix excluding the taxa that can be safely removed.
removed.matrix	A character-taxon matrix of the taxa that can be safely removed.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Wilkinson, M., 1995. Coping with abundant missing entries in phylogenetic inference using parsimony. Systematic Biology, 44, 501-514.

Examples

```
# Performs STR on the Gauthier 1986 dataset used in Wilkinson (1995):
str.out <- SafeTaxonomicReduction(Gauthier1986)

# View deleted taxa:
str.out$str.list

# View reduced matrix:
str.out$reduced.matrix

# View removed matrix:
str.out$removed.matrix
```

SafeTaxonomicReinsertion

Reinsert Safely Removed Taxa Into A Tree

Description

Following Safe Taxonomic Reduction reinsert removed taxa.

Usage

```
SafeTaxonomicReinsertion(treefile.in, treefile.out, str.list,
  multi.placements = "exclude")
```

Arguments

`treefile.in`	A Newick-formatted tree file containing tree(s) without safely removed taxa.
`treefile.out`	A file name where the newly generated trees will be written out to (required).
`str.list`	The safe taxonomic reduction table as generated by SafeTaxonomicReduction.
`multi.placements`	An optional for what to do with taxa that have more than one possible reinsertion position. Options are `"exclude"` (does not reinsert them; the default) or `"random"` (picks one of the possible positions and uses that; will vary stochastically if multiple trees exist).

Details

TEXT NEEDED!

Value

Nothing is returned, but a new file (`treefile.out`) is written.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Nothing yet
```

`StackPlot` *Plot Stacked Ordination Spaces*

Description

Plots a stack of ordination spaces representing multiple time-slices.

Usage

```
StackPlot(ordination_axes, ages, groups = NULL, time_slices,
  shear = 0.2, x_axis = 1, y_axis = 2, axis_label = "PC")
```

Arguments

`ordination_axes`	A matrix of the ordination axes supplied (rownames should be object names). First column should be values for first axis, second for second axis and so on.
`ages`	A two-column matrix of the first and last apperance dates for the taxa in the same format supplied to DatePhylo.
`groups`	A vector of colours (for plotting) with for each object name.
`time_slices`	A vector of the boundaries for a series of time slices.
`shear`	A single value (0 to 1) for the degree of shearing in the stacked ordination spaces.
`x_axis`	The ordination axis to plot on the x-axis.
`y_axis`	The ordination axis to plot nn the y-axis.
`axis_label`	The text used to precede the axis number. Here "PC" (for principal components/coordinates) is the assumed default, but the user may wish to use something else like "RW" instead.

Details

This style of plot is taken from various papers by Michael Foote (Foote 1993; his Figures 2, 4, 6, 8, 10, 12, and 14; Foote 1994; his Figure 2; Foote 1995; his Figure 3; Foote 1999; his Figure 22), and can be seen elsehwere in the litarture (e.g., Friedman and Coates 2006; their Figure 2c).

Here multiple ordination (or morpho-) spaces are plotted as a seriours of successive "stacks" representing specific intervals of time. Following geologic conventions the oldest time-slice is plotted at the base and the sequence gets younger towards the top.

Note that the user needs to supply three pieces of data: 1) a matrix representing the ordination axes (NB: these can come from any source, they do not have to be from Claddis functions), 2) a set of ages (first adn last appearances) in the same format as required by the DatePhylo function in the strap library, and 3) a vector of ages marking the boundaries of the time-slices.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com> and Emma Sherratt <emma.sherratt@gmail.com>

References

Foote, M., 1993. Discordance and concordance between morphological and taxonomic diversity. Paleobiology, 19, 185-204.

Foote, M., 1994. Morphological disparity in Ordovician-Devonian crinoids and the early saturation of morphological space. Paleobiology, 20, 320-344.

Foote, M., 1995. Morphological diversification of Paleozoic crinoids. Paleobiology, 21, 273-299.

Foote, M., 1999. Morphological diversity in the evolutionary radiation of Paleozoic and post-Paleozoic crinoids. Paleobiology, 25, 1-115.

Friedman, M. and Coates, M. I., 2006. A newly recognized fossil coelacanth highlights the early morphological diversification of the clade. Proceedings of the Royal Society of London B, 273, 245-250

Examples

```
# Create x-values that will form a grid:
x <- c(c(seq(0, 100, length.out = 101), seq(0, 100, length.out = 101),
  seq(0, 100, length.out = 101), seq(0, 100, length.out = 101)),
  c(rep(20, 101), rep(40, 101), rep(60, 101), rep(80, 101)))

# Create y-values that will form grid:
y <- c(c(rep(20, 101), rep(40, 101), rep(60, 101), rep(80, 101)),
  c(seq(0, 100, length.out = 101), seq(0, 100, length.out = 101),
  seq(0, 100, length.out = 101), seq(0, 100, length.out = 101)))

# Combine x and y values into
ordination_axes <- matrix(c(x, y), ncol = 2, dimnames =
  list(as.list(apply(matrix(sample(LETTERS, 8 * 8 * 101,
  replace = TRUE), nrow = 8 * 101), 1, paste, collapse = "")), NULL))

# Assign ages as though taxa range through entire interval (100-0 Ma):
```

```
ages <- matrix(c(rep(100, 8 * 101), rep(0, 8 * 101)), ncol = 2,
  dimnames = list(as.list(rownames(ordination_axes)), as.list(c("FAD",
  "LAD"))))

# Create five 20 million year time slices:
time_slices <- seq(0, 100, length.out = 6)

# Plot grid lines to show "shearing" effect is working:
StackPlot(ordination_axes = ordination_axes, ages = ages, time_slices = time_slices)

# Set random seed:
set.seed(17)

# Create random values to represent ordination axes:
ordination_axes <- matrix(rnorm(10000), nrow = 100, dimnames =
  list(as.list(apply(matrix(sample(LETTERS, 8 * 100, replace = TRUE), nrow = 100),
  1, paste, collapse = "")), NULL))

# Create random first and last appearance dates for objects:
ages <- matrix(as.vector(apply(matrix(runif(200, 0, 100), ncol = 2), 1, sort,
  decreasing = TRUE)), ncol = 2, byrow=TRUE, dimnames =
  list(as.list(rownames(ordination_axes)), as.list(c("FAD", "LAD"))))

# Create five 20 million year long time slices:
time_slices <- seq(0, 100, length.out = 6)

# Define groups for objects at random ("red" and "blue"):
groups <- sample(x = c("red", "blue"), size = nrow(ordination_axes), replace = TRUE)

# Randomly assign objects to groups:
names(groups) <- rownames(ordination_axes)

# Make stacked ordination plot with convex hulls for groups:
StackPlot(ordination_axes, ages, groups, time_slices)
```

`TrimMorphDistMatrix` *Trims a morphological distance matrix*

Description

Trims a morphological distance matrix by removing objects that cause empty cells.

Usage

```
TrimMorphDistMatrix(dist.matrix, Tree = NULL)
```

Arguments

dist.matrix	A distance matrix in the format created by MorphDistMatrix.
Tree	If the distance matrix includes ancestors this should be the tree (phylo object) used to reconstruct them.

Details

Trims a morphological distance matrix by removing nodes (terminal or internal) that cause empty cells allowing it to be passed to an ordination function such as cmdscale.

Some distances are not calculable from cladistic matrices if there are taxa that have no coded characters in common. This algorithm iteratively removes the taxa responsible for the most empty cells until the matrix is complete (no empty cells).

If the matrix includes reconstructed ancestors the user should also provide the tree used (as the tree argument). The function will then also remove the tips from the tree and where reconstructed ancestors also cause empty cells will prune the minimum number of descendants of that node. The function will then renumber the nodes in the distance matrix so they match the pruned tree.

Value

DistMatrix	A complete distance matrix with all cells filled. If there were no empty cells will return original.
Tree	A tree (if supplied) with the removed taxa (see below) pruned. If no taxa are dropped will return the same tree as inputted. If no tree is supplied this is set to NULL.
RemovedTaxa	A character vector listing the taxa removed. If none are removed this will be set to NULL.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Get morphological distances for Michaux (1989) data set:
distances <- MorphDistMatrix(Michaux1989)

# Attempt to trim max.dist.matrix:
TrimMorphDistMatrix(distances$DistanceMatrix)
```

`WriteMorphNexus` *Writes out a morphological #NEXUS data file*

Description

Writes out a morphological data file in #NEXUS format.

Usage

```
WriteMorphNexus(CladisticMatrix, filename)
```

Arguments

`CladisticMatrix`	The cladistic matrix in the format imported by ReadMorphNexus.
`filename`	The file name to write to. Should end in `.nex`.

Details

Writes out a #NEXUS (Maddison et al. 1997) data file representing the distribution of characters in a set of taxa. Data must be in the format created by importing data with ReadMorphNexus.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P., 1997. NEXUS: an extensible file format for systematic information. Systematic Biology, 46, 590-621.

See Also

WriteMorphTNT

Examples

```
# Write out Michaux 1989 to current working directory:
WriteMorphNexus(CladisticMatrix = Michaux1989, filename = "Michaux1989.nex")

# Remove file when finished:
file.remove("Michaux1989.nex")
```

`WriteMorphTNT` *Writes out a morphological TNT data file*

Description

Writes out a morphological data file in Hennig86/TNT format.

Usage

```
WriteMorphTNT(CladisticMatrix, filename, add.analysis.block = FALSE)
```

Arguments

`CladisticMatrix`	The cladistic matrix in the format imported by ReadMorphNexus.
`filename`	The file name to write to. Should end in `.tnt`.
`add.analysis.block`	Whether or not to add analysis block (i.e., tree search commands).

Details

Writes out a TNT (Goloboff et al. 2008) data file representing the distribution of discrete morphological characters in a set of taxa. Data must be in the format created by importing data with ReadMorphNexus.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Goloboff, P., Farris, J. and Nixon, K., 2008. TNT, a free program for phylogenetic analysis. Cladistics, 24, 774-786.

See Also

WriteMorphNexus

Examples

```
# Write out Michaux 1989 to current working directory:
WriteMorphTNT(CladisticMatrix = Michaux1989, filename = "Michaux1989.tnt")

# Remove file when finished:
file.remove("Michaux1989.tnt")
```

Index