Package 'Claddis'

September 26, 2020

Type Package

Title Measuring Morphological Diversity and Evolutionary Tempo

Version 0.6.3

Date 2020-09-14

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

Depends ape, phytools, strap, R (>= 3.5.0)

Imports clipr, geoscale, graphics, grDevices, methods, stats, utils

Suggests rgl, testthat

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 read_nexus_matrix(), and writes to both #NEXUS and TNT format (Goloboff et al. (2008) <doi:10.1111/j.1096-0031.2008.00217.x>). Main functions are test_rates(), which implements AIC and 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 MatrixDistances(), 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>. This also includes the GED correction from Lehmann et al. (2019) <doi:10.1111/pala.12430>. Multiple functions implement morphospace plots: plot_chronophylomorphospace() implements Sakamoto and Ruta (2012) <doi:10.1371/journal.pone.0039752>, plot_morphospace() implements Wills et al. (1994) <doi:10.1017/S009483730001263X>, plot_changes_on_tree() implements Wang and Lloyd (2016) <doi:10.1098/rspb.2016.0214>, and plot_morphospace_stack() implements Foote (1993) <doi:10.1017/S0094837300015864>. Other functions include safe_taxonomic_reduction(), which implements Wilkinson (1995) <doi:10.1093/sysbio/44.4.501>, map_dollo_changes() implements the Dollo stochastic character mapping of Tarver et al. (2018) <doi:10.1093/gbe/evy096>, and estimate_ancestral_states() implements

the ancestral state options of Lloyd (2018) <doi:10.1111/pala.12380>.

Encoding UTF-8

License GPL (>= 2)

LazyData yes

ByteCompile yes

RoxygenNote 7.1.0

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 2020-09-26 04:30:23 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 <- calculate_morphological_distances(cladistic_matrix = michaux_1989)

# Show distances:
distances
```

`align_matrix_block` *Aligns a phylogenetic matrix block*

Description

Given a block of taxa and characters aligns text so each character block begins at same point.

Usage

```
align_matrix_block(matrix_block)
```

Arguments

`matrix_block`	The matrix block as raw input text.

Details

The function serves to help build NEXUS files by neatly aligning raw text blocks of taxa and characters. Or in simple terms it takes input that looks like this:

```
Allosaurus  012100?1011
Abelisaurus  0100???0000
Tyrannosaurus  01012012010
Yi  10101?0????
```

And turns it into something that looks like this:

```
Allosaurus     012100?1011
Abelisaurus    0100???0000
Tyrannosaurus  01012012010
Yi             10101?0????
```

I use this in building the NEXUS files on my site, graemetlloyd.com.

Value

Nothing is returned, instead the aligned block is sent to the clipboard ready for pasting into a text editor.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Build example block from above:
x <- paste(c(
  "Allosaurus  012100?1011",
  "Abelisaurus  0100???0000",
  "Tyrannosaurus  01012012010",
  "Yi  10101?0????"
), collapse = "\n")

# Look at block pre-alignment:
x

# Align block and place on clipboard:
## Not run:
align_matrix_block(x)

## End(Not run)

# To test the response open a text editor and paste the
# contents of the clipboard.
```

`assign_taxa_to_bins` *Assign taxa to time bins*

Description

Given a set of first and last appearances assigns a set of taxa to a series of time bins.

Usage

```
assign_taxa_to_bins(taxon_ages, named_time_bins)
```

Arguments

`taxon_ages`	A matrix of taxon ages, with columns for first (`"fad"`) and last (`"lad"`) appearances and rownames correspodning to taxon names.
`named_time_bins`	A similar matrix of time bins, with columns for bottom (`"fad"`) and top (`"lad"`) ages and rownames corresponding to time bin names (e.g., geologic stages or other unit names).

Details

The various disparity plotting functions (plot_chronophylomorphospace, plot_morphospace_stack, plot_morphospace, plot_multi_morphospace) are designed to allow assignment of taxa to named groups so that these groups may be assigned different colours when plotting. One way taxa may be grouped is temporally, by assignment to a series of time bins.

There are many ways this may be automated and this function provides a very simple one: if the first and last appearance dates of a taxon overlap with a time bin then it can be assigned to that time bin. (In practice, taxa often have multiple occurrences with "ranges" that really represent uncertainty around their true age.)

Note that it is recommended that time bins be named without special characters beyond letters and underscores.

Value

A named list of taxa assigned to time bins.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

plot_chronophylomorphospace, plot_morphospace_stack, plot_morphospace, plot_multi_morphospace, ordinate_cladistic_matrix

Examples

```
# Build example named time bins:
named_time_bins <- matrix(data = c(443.8, 358.9, 358.9, 298.9, 298.9, 251.9,
  251.9, 201.3, 201.3, 145.0, 145.0, 65.5, 65.5, 23.03), ncol = 2,
  byrow = TRUE, dimnames = list(c("Silurodevonian", "Carboniferous",
  "Permian", "Triassic", "Jurassic", "Cretaceous", "Paleogene"),
  c("fad", "lad")))

# Build example taxon ages:
taxon_ages <- matrix(data = c(385.3, 374.5, 407, 374.5, 251, 228, 385.3,
  251, 251, 251, 391.8, 251, 251, 228, 385.3, 391.8, 391.8, 385.3, 311.7,
  359.2, 359.2, 416, 407, 407, 407, 407, 385.3, 397.5, 385.3, 161.2, 385.3,
  345.3, 318.1, 385.3, 228, 385.3, 385.3, 385.3, 385.3, 385.3, 385.3, 385.3,
  385.3, 385.3, 391.8, 407, 391.8, 374.5, 407, 70.6, 311.7, 407, 145.5, 251,
  65.5, 251, 112, 374.5, 374.5, 374.5, 385.3, 311.7, 249.7, 359.2, 391.8,
  374.5, 385.3, 83.5, 418.7, 251, 385.3, 391.8, 374.5, 345.3, 385.3, 385.3,
  407, 411.2, 397.5, 345.3, 374.5, 407, 216.5, 326.4, 411.2, 411.2, 374.5,
  359.2, 391.8, 359.2, 245, 216.5, 374.5, 245, 245, 245, 385.3, 245, 245,
  199.6, 374.5, 385.3, 385.3, 374.5, 306.5, 345.3, 345.3, 411.2, 397.5,
  397.5, 397.5, 397.5, 374.5, 391.8, 374.5, 145.5, 374.5, 326.4, 311.7,
  374.5, 199.6, 374.5, 374.5, 374.5, 374.5, 374.5, 374.5, 374.5, 374.5,
  374.5, 385.3, 397.5, 385.3, 359.2, 397.5, 65.5, 306.5, 397.5, 99.6, 245,
  23.03, 245, 99.6, 359.2, 359.2, 359.2, 374.5, 306.5, 247.4, 318.1, 385.3,
  359.2, 374.5, 70.6, 416, 250.4, 374.5, 385.3, 359.2, 326.4, 374.5, 374.5,
  397.5, 407, 391.8, 326.4, 359.2, 397.5, 203.6, 318.1, 407, 407),
  ncol = 2, dimnames = list(c("Adololopas_moyasmithae", "Adelargo_schultzei",
  "Amadeodipterus_kencampbelli", "Andreyevichthys_epitomus",
  "Aphelodus_anapes", "Archaeoceratodus_avus", "Archaeonectes_pertusus",
  "Arganodus_atlantis", "Ariguna_formosa", "Asiatoceratodus_sharovi",
  "Barwickia_downunda", "Beltanodus_ambilobensis", "Ceratodus_formosa",
```

```
  "Ceratodus_latissimus", "Chirodipterus_australis",
  "Chirodipterus_onawwayensis", "Chirodipterus_rhenanus",
  "Chirodipterus_wildungensis", "Conchopoma_gadiforme", "Ctenodus_romeri",
  "Delatitia_breviceps", "Diabolepis_speratus", "Dipnorhynch_cathlesae",
  "Dipnorhynchus_sussmilchi", "Dipnorhynchus_kiandrensis",
  "Dipnorhynchus_kurikae", "Dipterus_cf_valenciennesi",
  "Dipterus_valenciennesi", "Eoctenodus_microsoma",
  "Ferganoceratodus_jurassicus", "Fleurantia_denticulata",
  "Ganopristodus_splendens", "Gnathorhiza_serrata", "Gogodipterus_paddyensis",
  "Gosfordia_truncata", "Griphognathus_minutidens", "Griphognathus_sculpta",
  "Griphognathus_whitei", "Grossipterus_crassus", "Holodipterus_elderae",
  "Holodipterus_gogoensis", "Robinsondipterus_longi",
  "Asthenorhynchus_meemannae", "Holodipterus_santacrucensis",
  "Howidipterus_donnae", "Ichnomylax_kurnai", "Iowadipterus_halli",
  "Jarvikia_arctica", "Jessenia_concentrica", "Lepidosiren_paradoxa",
  "Megapleuron_zangerli", "Melanognathus_canadensis",
  "Metaceratodus_wollastoni", "Microceratodus_angolensis",
  "Mioceratodus_gregoryi", "Namatozodia_pitikanta", "Neoceratodus_forsteri",
  "Nielsenia_nordica", "Oervigia_nordica", "Orlovichthys_limnatis",
  "Palaeodaphus_insignis", "Palaeophichthys_parvulus", "Paraceratodus_germaini",
  "Parasagenodus_sibiricus", "Pentlandia_macroptera",
  "Phaneropleuron_andersoni", "Pillararhynchus_longi", "Protopterus_annectens",
  "Psarolepis_romeri", "Ptychoceratodus_serratus", "Rhinodipterus_secans",
  "Rhinodipterus_ulrichi", "Rhynchodipterus_elginensis", "Sagenodus_inaequalis",
  "Scaumenacia_curta", "Soederberghia_groenlandica",
  "Sorbitorhynchus_deleaskitus", "Speonesydrion_iani", "Stomiahykus_thlaodus",
  "Straitonia_waterstoni", "Sunwapta_grandiceps", "Tarachomylax_oepiki",
  "Tellerodus_sturi", "Tranodis_castrensis", "Uranolophus_wyomingensis",
  "Westollrhynchus_lehmanni"), c("fad", "lad")))

# Assign taxa to time bins:
assign_taxa_to_bins(taxon_ages = taxon_ages, named_time_bins = named_time_bins)
```

`bin_changes` *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

```
bin_changes(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 (between 100 and 0 Ma):
change_times <- stats::runif(n = 100, min = 0, max = 100)

# Create 10 equal-length time bins:
time_bins <- seq(100, 0, length.out = 11)

# Get N changes for each bin:
bin_changes(change_times, time_bins)
```

---

bin_character_completeness

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

```
bin_character_completeness(
  cladistic_matrix,
  time_tree,
  time_bins,
  plot = FALSE,
  confidence.interval = 0.95
)
```

Arguments

`cladistic_matrix`	A cladistic matrix in the form imported by read_nexus_matrix.
`time_tree`	A time-scaled phylogenetic tree containing all the taxa in `cladistic_matrix`.
`time_bins`	A set of time bin boundaries (oldest to youngest) in millions of years.
`plot`	An optional choice to plot the results (default is `FALSE`).
`confidence.interval`	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 confidence interval, and per character proportional character completeness in each time bin.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random tree for the Day et al. 2016 data set:
day_2016tree <- ape::rtree(n = nrow(day_2016$matrix_1$matrix))
day_2016tree$tip.label <- rownames(x = day_2016$matrix_1$matrix)
day_2016tree$root.time <- max(diag(x = ape::vcv(phy = day_2016tree)))

# Get proportional phylogenetic character completeness in ten equal-length
# time bins:
bin_character_completeness(
  cladistic_matrix = day_2016,
  time_tree = day_2016tree, time_bins = seq(
    from =
      day_2016tree$root.time, to = day_2016tree$root.time -
      max(diag(x = ape::vcv(phy = day_2016tree))), length.out = 11
  )
)

# Same, but with a plot:
bin_character_completeness(
  cladistic_matrix = day_2016,
  time_tree = day_2016tree, time_bins = seq(
```

```
    from =
      day_2016tree$root.time, to = day_2016tree$root.time -
      max(diag(x = ape::vcv(phy = day_2016tree))), length.out = 11
  ), plot = TRUE
)
```

`bin_edge_lengths` *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

```
bin_edge_lengths(time_tree, time_bins, pruned_tree = NULL)
```

Arguments

`time_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 `time_tree`.

Details

Calculates the total edge duration of a time-scaled tree present in 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

`binned_edge_lengths`
: 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.

`binned_terminal_edge_lengths`
: As above, but counting terminal edges only.

`binned_internal_edge_lengths`
: As above, but counting internal edges only.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a random 10-taxon tree:
time_tree <- ape::rtree(n = 10)

# Add root age:
time_tree$root.time <- max(diag(ape::vcv(time_tree)))

# Create time bins:
time_bins <- seq(from = time_tree$root.time, to = 0, length.out = 11)

# Get edge lengths for each bin:
bin_edge_lengths(time_tree = time_tree, time_bins = time_bins)
```

---

build_cladistic_matrix

Creates a morphological data file from a matrix

Description

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

Usage

```
build_cladistic_matrix(
  character_taxon_matrix,
  header = "",
  character_weights = NULL,
  ordering = NULL,
  symbols = NULL,
  equalise.weights = FALSE,
  ignore_duplicate_taxa = FALSE
)
```

Arguments

character_taxon_matrix	A Character-Taxon (columns-rows) matrix, with taxon names as rownames.
header	A scalar indicating any header text (defaults to an empty string: "").
character_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.

`ignore_duplicate_taxa`
: Logical indicating whether or not to ignore (allow; TRUE) duplicate taxa or not (FALSE; default).

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

compactify_matrix, prune_cladistic_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_nexus_matrix, write_tnt_matrix

Examples

```
# Create random 10-by-50 matrix:
character_taxon_matrix <- 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:
build_cladistic_matrix(character_taxon_matrix)
```

`calculate_morphological_distances` *Get distance matrices from a cladistic matrix*

Description

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

Usage

```
calculate_morphological_distances(
  cladistic_matrix,
  distance_metric = "mord",
  ged_type = "wills",
  distance_transformation = "arcsine_sqrt",
  polymorphism_behaviour = "min_difference",
  uncertainty_behaviour = "min_difference",
  inapplicable_behaviour = "missing",
  character_dependencies = NULL,
  alpha = 0.5
)
```

Arguments

`cladistic_matrix`
: A character-taxon matrix in the format imported by read_nexus_matrix.

`distance_metric`
: The distance metric to use. Must be one of `"gc"`, `"ged"`, `"red"`, or `"mord"` (the default).

`ged_type`
: The type of GED to use. Must be one of `"legacy"`, `"hybrid"`, or `"wills"` (the default). See details for an explanation.

`distance_transformation`
: The type of distance transformation to perform. Options are `"none"`, `"sqrt"`, or `"arcsine_sqrt"` (the default). (Note: this is only really appropriate for the proportional distances, i.e., "gc" and "mord".)

`polymorphism_behaviour`
: The distance behaviour for dealing with polymorphisms. Must be one of `"mean_difference"`, `"min_difference"` (the default), or `"random"`.

`uncertainty_behaviour`
: The distance behaviour for dealing with uncertainties. Must be one of `"mean_difference"`, `"min_difference"` (the default), or `"random"`.

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

`character_dependencies`
: Only relevant if using `inapplicable_behaviour = "hsj"`. Must be a two-column matrix with colnames "dependent_character" and "independent_character" that specifies character hierarchies. See details.

`alpha`
: The alpha value (sensu Hopkins and St John 2018). Only relevant if using `inapplicable_behaviour = "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 `ged_type`, with `"wills"` being the WIlls (1998) implementation. An advantage of this misinterpreted form of the 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 `ged_type` - (`"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 (`distance_transformation` 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 `polymorphism_behaviour` and `uncertainty_behaviour`). 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. Currently 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 `character_dependencies` option. This must be in the form of a two-column matrix with column headers of `"dependent_character"` and `"independent_character"`. 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 `inapplicable_behaviour = "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 between 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_metric` option). Addition-

ally, 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 `character_dependencies` option. Furthermore, because of this any level of dependency is possible (if unlikely), e.g., quarternary etc.

Value

`distance_metric`
: The distance metric used.

`distance_matrix`
: The pairwise distance matrix generated.

`comparable_character_matrix`
: 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. *Biometrika*, **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 <- calculate_morphological_distances(cladistic_matrix = day_2016)

# Show distance metric:
distances$distance_metric

# Show distance matrix:
distances$distance_matrix

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

```
# 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):
character_dependencies <- matrix(c(8, 7),
  ncol = 2,
  byrow = TRUE, dimnames = list(
    c(),
    c(
      "dependent_character",
      "independent_character"
    )
  )
)

# Get morphological distances for the Day et
# al. (2016) data set using HSJ approach:
distances <- calculate_morphological_distances(
  cladistic_matrix = day_2016,
  inapplicable_behaviour = "hsj",
  character_dependencies = character_dependencies,
  alpha = 0.5
)

# Show distance metric:
distances$distance_metric

# Show distance matrix:
distances$distance_matrix

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

`compactify_matrix` *Collapses matrix to unique character state distributions*

Description

Collapses a cladistic matrix to just unique character state distributions and taxon names.

Usage

```
compactify_matrix(cladistic_matrix, message = TRUE)
```

Arguments

`cladistic_matrix`
: The cladistic matrix in the format imported by read_nexus_matrix.

`message` Logical indicating whether or not a message should be printed to the screen if the matrix cannot be compactified.

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 and will merge duplicate taxon names into unique character strings.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

build_cladistic_matrix, prune_cladistic_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_nexus_matrix, write_tnt_matrix

Examples

```
# Examine the matrix pre-compactification:
michaux_1989$matrix_1$matrix

# Examine the weights pre-compactification:
michaux_1989$matrix_1$character_weights

# Compactify the matrix:
michaux_1989compact <- compactify_matrix(michaux_1989)

# Examine the matrix post-compactification:
michaux_1989compact$matrix_1$matrix

# Examine the weights post-compactification:
michaux_1989compact$matrix_1$character_weights
```

---

date_nodes *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

```
date_nodes(time_tree)
```

Arguments

time_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:
time_tree <- ape::read.tree(text = "(A:1,(B:1,(C:1,D:1):1):1);")

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

# Get node ages:
date_nodes(time_tree = time_tree)
```

---

day_2016 *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 read_nexus_matrix.

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.

`estimate_ancestral_states` *Ancestral Character State Estimation*

Description

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

Usage

```
estimate_ancestral_states(
  cladistic_matrix,
  time_tree,
  estimate_all_nodes = FALSE,
  estimate_tip_values = FALSE,
  inapplicables_as_missing = FALSE,
  polymorphism_behaviour = "equalp",
  uncertainty_behaviour = "equalp",
  threshold = 0.01,
  all_missing_allowed = FALSE
)
```

Arguments

`cladistic_matrix`	A character-taxon matrix in the format imported by read_nexus_matrix.
`time_tree`	A tree (phylo object) with branch lengths that represents the relationships of the taxa in `cladistic_matrix`.
`estimate_all_nodes`	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).
`estimate_tip_values`	Logical that allows the user to make estimates for tip values. The default (`FALSE`) will only makes estimates for internal nodes (recommended).
`inapplicables_as_missing`	Logical that decides whether or not to treat inapplicables as missing (TRUE) or not (FALSE, the default and recommended option).
`polymorphism_behaviour`	One of either "equalp" or "treatasmissing".
`uncertainty_behaviour`	One of either "equalp" or "treatasmissing".
`threshold`	The threshold value to use when collapsing marginal likelihoods to discrete state(s).
`all_missing_allowed`	Logical to allow all missing character values (generally not recommended, hence default is FALSE).

Details

At its' core the function uses either the rerootingMethod (Yang et al. 1995) as implemented in the phytools package (for discrete characters) or the ace function in the ape package (for 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 developed for Brusatte et al. (2014).

The function can deal with ordered or unordered characters and does so by allowing only indirect transitions (from 0 to 2 must pass through 1) or direct transitions (from 0 straight to 2), respectively. However, more complex step matrix transitions are not currently supported.

Ancestral state estimation is complicated where polymorphic or uncertain tip values exist. These are not currently well handled here, although see the `fitpolyMk` function in phytools for a way these could be dealt with in future. The only available options right now are to either treat multiple states as being equally probable of the "true" tip state (i.e., a uniform prior) or to avoid dealing with them completely by treating them as missing (NA) values.

It is also possible to try to use phylogenetic information to infer missing states, both for internal nodes (e.g., those leading to missing tip states) and for tips. This is captured by the `estimate_all_nodes` and `estimate_tip_values` options. These have been partially explored by Lloyd (2018), who cautioned against their use.

Value

The function will return the same `cladistic_matrix`, but with two key additions: 1. Internal nodes (numbered by ape formatting) will appear after taxa in each matrix block with estimated states coded for them, and 2. The time-scaled tree used will be added to `cladistic_matrix` as `cladistic_matrix$topper$tree`. Note that if using the `estimate_tip_values = TRUE` option then tip values may also be changed from those provided as input.

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.

Lloyd, G. T., 2018. Journeys through discrete-character morphospace: synthesizing phylogeny, tempo, and disparity. *Palaeontology*, **61**, 637-645.

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:
time_tree <- ape::rtree(n = nrow(day_2016$matrix_1$matrix))

# Update taxon names to match those in the data matrix:
time_tree$tip.label <- rownames(x = day_2016$matrix_1$matrix)

# Set root time by making youngest taxon extant:
time_tree$root.time <- max(diag(x = ape::vcv(phy = time_tree)))

# Use Day matrix as cladistic matrix:
cladistic_matrix <- day_2016

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

# Estimate ancestral states:
estimate_ancestral_states(
  cladistic_matrix = cladistic_matrix,
  time_tree = time_tree
)
```

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

Description

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

Usage

```
find_descendant_edges(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 <- ape::read.tree(text = "(A,(B,(C,D)));")

# Plot tree:
plot(tree)

# Show nodelabels:
nodelabels()

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

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

---

find_linked_edges *Find linked edges for a tree*

Description

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

Usage

```
find_linked_edges(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 <- ape::read.tree(text = "(A,(B,(C,D)));")

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

`find_minimum_spanning_edges` *Get edges of minimum spanning tree*

Description

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

Usage

```
find_minimum_spanning_edges(distance_matrix)
```

Arguments

`distance_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:
distance_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:
distance_matrix

# Use find_minimum_spanning_edges to get the edges for the minimum spanning
# tree:
find_minimum_spanning_edges(distance_matrix)

# Use sum of find_minimum_spanning_edges to get the length of the minimum
# spanning tree:
sum(find_minimum_spanning_edges(distance_matrix))
```

`find_mrca` *Find ancestor*

Description

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

Usage

```
find_mrca(descendant_names, tree)
```

Arguments

`descendant_names`	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 trim_matrix, but potentially of more general use.

Value

`ancestor_node`	The ancestral node number.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

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

# Plot the tree:
ape::plot.phylo(tree)
```

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

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

`fix_root_time` *Fixes root.time after taxa have been pruned from a tree*

Description

Fixes root.time after taxa have been pruned from a tree using ape::drop.tip

Usage

```
fix_root_time(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 fixed and that is what this function does.

Note that `fix_root_time` 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 fixed `$root.time`.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Create a simple four-taxon tree with branch lengths:
tree <- ape::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 <- ape::drop.tip(phy = tree, tip = "A")

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

# Use the function to fix the root time:
pruned_tree <- fix_root_time(original_tree = tree, pruned_tree = pruned_tree)

# Show that the root time is now fixed (19 Ma):
pruned_tree$root.time
```

---

gauthier_1986 *Character-taxon matrix from Gauthier 1986*

Description

The character-taxon matrix from Gauthier (1986).

Format

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

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.

---

map_dollo_changes *Stochastic Character Map For Dollo Character*

Description

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

Usage

```
map_dollo_changes(time_tree, tip_states)
```

Arguments

time_tree	A tree in phylo format with positive branch lengths and a value for $root.time.
tip_states	A named vector of tip states (must be 0 or 1), where the names match tree$tip.label.

Details

The non-ideal solution from Tarver et al. (2018) 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 (1 -> 0).

The function operates as follows:

1) Establishes the least inclusive clade exhibiting the derived state (1). 2) Assumes a single gain occurred with equal probability along the branch subtending this clade. 3) Prunes the inclusive clade to generate a subtree with a strong root prior 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 characterise this uncertainty.

Value

changes	A matrix of all changes (gains and losses).
stochastic_character_map	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

```
# Build example ten-tip tree:
time_tree <- ape::read.tree(text = paste0("(A:1,(B:1,((C:1,(D:1,(E:1,F:1):1):1):1,",
  "((G:1,H:1):1,(I:1,J:1):1):1):1):1);"))

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

# Build example tip state values:
tip_states <- c(A = 0, B = 0, C = 1, D = 1, E = 0, F = 1, G = 1, H = 1, I = 0, J = 1)

# Run map_dollo_changes on data and store output:
```

```
out <- map_dollo_changes(time_tree, tip_states)

# View matrix of changes:
out$changes

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

`map_stochastic_changes` *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

```
map_stochastic_changes(
  cladistic_matrix,
  time_tree,
  time_bins,
  n_simulations = 10,
  polymorphism_behaviour = "equalp",
  uncertainty_behaviour = "equalp",
  inapplicable_behaviour = "missing"
)
```

Arguments

`cladistic_matrix`	A character-taxon matrix in the format imported by read_nexus_matrix.
`time_tree`	A time-scaled tree (phylo object) that represents the relationships of the taxa in `cladistic_matrix`.
`time_bins`	A vector of ages representing the boundaries of a series of time bins.
`n_simulations`	The number of simulations to perform (passed to make.simmap.
`polymorphism_behaviour`	What to do with polymorphic (&) characters. One of "equalp", "missing", or "random". See details.
`uncertainty_behaviour`	What to do with uncertain (/) characters. One of "equalp", "missing", or "random". See details.
`inapplicable_behaviour`	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

`all_state_changes`
: 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.

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

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

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

`binned_internal_edge_lengths`
: 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:
cladistic_matrix <- day_2016

# Prune out continuous characters:
cladistic_matrix <- prune_cladistic_matrix(
  cladistic_matrix =
    cladistic_matrix, blocks2prune = 1
)

# Prune out majority of characters so
# example runs quickly:
cladistic_matrix <- prune_cladistic_matrix(
  cladistic_matrix =
    cladistic_matrix, characters2prune = 1:32
)
```

```
# Generete random tree for matrix taxa:
time_tree <- ape::rtree(n = nrow(day_2016$matrix_1$matrix))

# Add taxon names to tree:
time_tree$tip.label <- rownames(x = day_2016$matrix_1$matrix)

# Add root age to tree:
time_tree$root.time <- max(diag(x = ape::vcv(phy = time_tree)))

# Get all state changes for two simulations:
state_changes <-
  map_stochastic_changes(
    cladistic_matrix = cladistic_matrix,
    time_tree = time_tree, time_bins = seq(time_tree$root.time, 0,
      length.out = 3
    ), n_simulations = 2
  )

# View matrix of all stochstic character changes:
state_changes$all_state_changes

# View vector of sampled time for each character:
state_changes$character_times

# View matrix of edge lengths in each time bin:
state_changes$binned_edge_lengths

# View matrix of terminal edge lengths in each time bin:
state_changes$binned_terminal_edge_lengths

# View matrix of internal edge lengths in each time bin:
state_changes$binned_internal_edge_lengths
```

`match_tree_edges` *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

```
match_tree_edges(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 <- ape::rtree(n = 10)

# Remove three leaves:
pruned_tree <- ape::drop.tip(phy = original_tree, tip = c("t1", "t3", "t8"))

# Find matching edges:
X <- match_tree_edges(original_tree, pruned_tree)

# Show matching edges:
X$matching_edges

# Show matching nodes:
X$matching_nodes

# Show removed edges:
X$removed_edges
```

---

michaux_1989 *Character-taxon matrix from Michaux 1989*

Description

The character-taxon matrix from Michaux (1989).

Format

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

References

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

`ordinate_cladistic_matrix` *Principal Coordinates on a Cladistic Matrix*

Description

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

Usage

```
ordinate_cladistic_matrix(
  cladistic_matrix,
  distance_metric = "mord",
  ged_type = "wills",
  distance_transformation = "arcsine_sqrt",
  distance_polymorphism_behaviour = "min_difference",
  distance_uncertainty_behaviour = "min_difference",
  distance_inapplicable_behaviour = "missing",
  character_dependencies = NULL,
  alpha = 0.5,
  correction = "cailliez",
  time_tree = NULL,
  estimate_all_nodes = FALSE,
  estimate_tip_values = FALSE,
  inapplicables_as_missing = FALSE,
  ancestral_polymorphism_behaviour = "equalp",
  ancestral_uncertainty_behaviour = "equalp",
  threshold = 0.01,
  all_missing_allowed = FALSE
)
```

Arguments

`cladistic_matrix` A character-taxon matrix in the format imported by read_nexus_matrix.

`distance_metric` See calculate_morphological_distances.

`ged_type` See calculate_morphological_distances.

`distance_transformation` See calculate_morphological_distances.

`distance_polymorphism_behaviour` See calculate_morphological_distances.

`distance_uncertainty_behaviour`
: See calculate_morphological_distances.

`distance_inapplicable_behaviour`
: See calculate_morphological_distances.

`character_dependencies`
: See calculate_morphological_distances.

`alpha`
: See calculate_morphological_distances.

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

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

`estimate_all_nodes`
: See estimate_ancestral_states.

`estimate_tip_values`
: See estimate_ancestral_states.

`inapplicables_as_missing`
: See estimate_ancestral_states.

`ancestral_polymorphism_behaviour`
: See estimate_ancestral_states.

`ancestral_uncertainty_behaviour`
: See estimate_ancestral_states.

`threshold`
: See estimate_ancestral_states.

`all_missing_allowed`
: See estimate_ancestral_states.

Details

Takes a cladistic matrix in the format imported by read_nexus_matrix and performs Principal Co-ordinates (Gower 1966) analysis on it.

This function is effectively a wrapper for the pipeline:

estimate_ancestral_states -> calculate_morphological_distances -> pcoa

With the first part being optional (if wanting a phylomorphospace) and the latter coming from the ape package (the user is referred there for some of the options, e.g., using the Caillez 1983 approach to avoiding negative eigenvalues). (See Lloyd 2016 for more on disparity pipelines.)

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 by the estimate_ancestral_states part.

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

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

Value

time_tree	The tree (if supplied). Note this may be pruned from the input tree by trim_matrix.
distance_matrix	The distance matrix. Note this may be pruned by trim_matrix and thus not include all taxa.
removed_taxa	A vector of taxa and/or nodes removed by trim_matrix. Returns NULL if none were removed.
note	See pcoa.
values	See pcoa.
vectors	See pcoa. Note: this will be the same as vectors.cor from the pcoa output if a correction was applied.
trace	See pcoa. Note: this will be the same as trace.cor from the pcoa output if a correction was applied.

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.

See Also

assign_taxa_to_bins, plot_chronophylomorphospace, plot_morphospace_stack, plot_morphospace, plot_multi_morphospace

Examples

```
# Run on Michaux (1989) data set with default settings:
x <- ordinate_cladistic_matrix(cladistic_matrix = michaux_1989)

# Show entire output:
x

# Generate a (made up) tree:
time_tree <- ape::rtree(n = length(x = rownames(x = michaux_1989$matrix_1$matrix)))

# Add taxon names to it:
time_tree$tip.label <- rownames(x = michaux_1989$matrix_1$matrix)

# Set root time by making youngest taxon extant:
time_tree$root.time <- max(diag(x = ape::vcv(phy = time_tree)))
```

```
# Run with tree:
y <- ordinate_cladistic_matrix(cladistic_matrix = michaux_1989, time_tree = time_tree)

# Show new output:
y
```

`partition_time_bins` *Time bin partitioner*

Description

Generates all possible contiguous partitions of N time bins.

Usage

```
partition_time_bins(n_time_bins, partition_sizes_to_include = "all")
```

Arguments

`n_time_bins`	The number of time bins.
`partition_sizes_to_include`	Either "all" (the default) or a vector of requested partition sizes.

Details

This function is designed for use with the test_rates function and generates all possible contiguous partitions of N time bins. This allows use of an information criterion like AIC to pick a "best" partition, weighing fit and partition number simultaneously.

You can also ask for only partitions of a specific number using the `partition_sizes_to_include` option. For example, `partition_sizes_to_include = c(1,2,3)` will only return partitions of 1, 2, or 3 sets of elements.

Value

Returns a list of lists of vectors ready for use in test_rates.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Get all partitions for four time bins:
partition_time_bins(n_time_bins = 4)

# Get all partitions for five time bins of size 2:
partition_time_bins(n_time_bins = 5, partition_sizes_to_include = 2)
```

plot_changes_on_tree *Plots character changes on branches*

Description

Plots character changes in boxes on branches.

Usage

```
plot_changes_on_tree(character_changes, time_tree, label_size = 0.5)
```

Arguments

character_changes	A matrix of character changes.
time_tree	Tree on which character changes occur.
label_size	The size of the text for the barnch labels. Default is 0.5.

Details

Takes the character_changes output from test_rates 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)

# Get first MPT for the Michaux data set:
time_tree <- ape::read.tree(text = paste0("(Ancilla:31.6,(Turrancilla:102.7,",
  "(Ancillista:1,Amalda:63.5):1):1);"))

# Set root time for tree:
time_tree$root.time <- 103.7

# Get discrete character rates (includes changes):
out <- test_rates(time_tree, michaux_1989,
  seq(time_tree$root.time, 0, length.out = 3),
  branch_partitions = list(list(1)), alpha = 0.01
)
```

```
# Plot character changes on the tree:
plot_changes_on_tree(
  out$inferred_character_changes,
  time_tree
)
```

`plot_chronophylomorphospace` *Chronophylomorphospace Plot*

Description

Plots a three-dimensional chronophylomorphospace.

Usage

```
plot_chronophylomorphospace(
  pcoa_input,
  x_axis = 1,
  y_axis = 2,
  taxon_groups = NULL,
  plot_tips = TRUE,
  plot_nodes = TRUE,
  plot_taxon_names = TRUE,
  plot_edges = TRUE,
  shadow = TRUE,
  plot_group_legend = TRUE,
  group_legend_position = "top_right",
  palette = "viridis"
)
```

Arguments

`pcoa_input`	Principal coordinate data in the format output by ordinate_cladistic_matrix 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.
`taxon_groups`	A named list of groups to which taxa are assigned (optional). This is used to plot points or convex hulls in different colours corresponding to each group. As the user names the groups these can represent any grouping of interest (e.g., taxonomic, ecological, temporal, spatial). assign_taxa_to_bins can automate temporal assignments.
`plot_tips`	Whether or not to plot the tip nodes (defaults to TRUE).
`plot_nodes`	Whether or not to plot the internal nodes (defaults to TRUE).
`plot_taxon_names`	Whether or not to show the taxon nodes (defaults to TRUE).

`plot_edges` Whether or not to plot the branches (defaults to TRUE).

`shadow` Whether or not to plot a shadow (2D plot) on the bottom face of the 3D plot (defaults to TRUE).

`plot_group_legend` Whether or not to add a legend to identify the groups. Only relevant if using `"taxon_groups"`.

`group_legend_position` Position to plot the group legend. Must be one of `bottom_left`, `bottom_right`, `top_left`, or `top_right` (the default).

`palette` The palette to use for plotting each element of taxon_groups. See palette.

Details

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

This function aims 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.

See Also

assign_taxa_to_bins, plot_morphospace_stack, plot_morphospace, plot_multi_morphospace, ordinate_cladistic_matrix

Examples

```
## Not run:
# Require rgl library to use:
require(rgl)

# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))
```

```
# Add root age to tree:
time_tree$root.time <- 269.5

# Prune incomplete taxa from tree:
time_tree <- ape::drop.tip(phy = time_tree, tip = c("Lycaenodon_longiceps",
  "Niuksenitia_sukhonensis"))

# Prune incomplete taxa from cladistic matrix:
cladistic_matrix <- prune_cladistic_matrix(cladistic_matrix = day_2016,
  taxa2prune = c("Lycaenodon_longiceps", "Niuksenitia_sukhonensis"))

# Perform a phylogenetic Principal Coordinates Analysis:
pcoa_input <- ordinate_cladistic_matrix(
  cladistic_matrix = cladistic_matrix,
  time_tree = time_tree
)

# Define some simple taxon groups for the data as a named list:
taxon_groups <- list(nonBurnetiamorpha = c("Biarmosuchus_tener",
  "Hipposaurus_boonstrai", "Bullacephalus_jacksoni", "Pachydectes_elsi",
  "Niuksenitia_sukhonensis", "Ictidorhinus_martinsi", "RC_20",
  "Herpetoskylax_hopsoni"),
  Burnetiamorpha = c("Lemurosaurus_pricei", "Lobalopex_mordax",
  "Lophorhinus_willodenensis", "Proburnetia_viatkensis", "Lende_chiweta",
  "Paraburnetia_sneeubergensis", "Burnetia_mirabilis", "BP_1_7098"))

# Plot a chronophylomorphospace:
plot_chronophylomorphospace(
  pcoa_input = pcoa_input,
  taxon_groups = taxon_groups,
)

## End(Not run)
```

`plot_morphospace` *Plot Morphopace*

Description

Plots a morphospace using the output from ordinate_cladistic_matrix.

Usage

```
plot_morphospace(
  pcoa_input,
  x_axis = 1,
  y_axis = 2,
  z_axis = NULL,
  taxon_groups = NULL,
```

```
  plot_taxon_names = FALSE,
  plot_convex_hulls = FALSE,
  plot_internal_nodes = FALSE,
  plot_edges = TRUE,
  plot_root = TRUE,
  root_colour = "red",
  palette = "viridis",
  plot_group_legend = TRUE,
  group_legend_position = "top_right",
  plot_z_legend = TRUE,
  z_legend_position = "bottom_right",
  inform = TRUE,
  x_limits = NULL,
  y_limits = NULL
)
```

Arguments

`pcoa_input`	The main input in the format output from ordinate_cladistic_matrix.
`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).
`taxon_groups`	A named list of groups to which taxa are assigned (optional). This is used to plot points or convex hulls in different colours corresponding to each group. As the user names the groups these can represent any grouping of interest (e.g., taxonomic, ecological, temporal, spatial). assign_taxa_to_bins can automate temporal assignments.
`plot_taxon_names`	Logical indicating whether to plot the names of the taxa (defaults to FALSE).
`plot_convex_hulls`	Logical indicating whether to plot convex hulls around any taxon_groups (if used).
`plot_internal_nodes`	Logical indicating whether to plot the internal nodes of the tree (if included in `pcoa_input`) (defaults to FALSE).
`plot_edges`	Logical indicating whether to plot the branches of the tree (if included in `pcoa_input`) (defaults to TRUE).
`plot_root`	Logical indicating whether to plot the root separately (defaults to FALSE).
`root_colour`	If plotting the root separately (previous option) sets the root colour.
`palette`	The palette to use for plotting each element of taxon_groups. See palette.
`plot_group_legend`	Logical indicating whether to plot a legend for taxon_groups. (Default is TRUE.)
`group_legend_position`	Position to plot the group legend. Must be one of `bottom_left`, `bottom_right`, `top_left`, or `top_right` (the default).

`plot_z_legend`	Logical indicating whether to plot a legend for the z-axis. (Default is TRUE.)
`z_legend_position`	Position to plot the group legend. Must be one of `bottom_left`, `bottom_right` (the default), `top_left`, or `top_right`.
`inform`	Logical indicating whether to inform the user of any taxon pruning. (Default is TRUE.)
`x_limits`	Plot limits to use for x-axis. Only intended for use by plot_multi_morphospace.
`y_limits`	Plot limits to use for y-axis. Only intended for use by plot_multi_morphospace.

Details

Uses output from ordinate_cladistic_matrix to make morphospace plots.

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

Will automatically generate phylomorphospaces if a tree was included in the ordination.

Can also plot groups of points - whether they represent taxonomic, ecological, temporal, or spatial groupings - in different colours as well as plot translucent convex hulls around these groups, by using the `taxon_groups` and `plot_convex_hulls = TRUE` options, respectively. Note that `taxon_groups` should be in the form of a named list (see example below for how these should be formatted).

Various other options allow toggling of particular features on or off. For example, the taxon names can be shown with `plot_taxon_names = TRUE`.

Note that some features will generate legends that may initially appear to disappear off the sides of the plot, but simple resizing of the plot window (or increasing the width:height ratio if outputting to a file) should fix this.

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.

See Also

assign_taxa_to_bins, plot_chronophylomorphospace, plot_morphospace_stack, plot_multi_morphospace, ordinate_cladistic_matrix

Examples

```
# Perform a PCoA ordination on the day_2016 data set:
pcoa_input <- ordinate_cladistic_matrix(cladistic_matrix = day_2016)

# Plot this as a simple bivarate morphospace:
plot_morphospace(pcoa_input = pcoa_input)

# Use the Wills technique to add a third axis (PC3):
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3)

# You may need to resize the plot to see the legend for the z-axis

# Add taxon names as well:
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3, plot_taxon_names = TRUE)

# Define some simple taxon groups for the data as a named list:
taxon_groups <- list(nonBurnetiamorpha = c("Biarmosuchus_tener",
  "Hipposaurus_boonstrai", "Bullacephalus_jacksoni", "Pachydectes_elsi",
  "Ictidorhinus_martinsi", "RC_20", "Herpetoskylax_hopsoni"),
  Burnetiamorpha = c("Lemurosaurus_pricei", "Lobalopex_mordax",
  "Lophorhinus_willodenensis", "Proburnetia_viatkensis", "Lende_chiweta",
  "Paraburnetia_sneeubergensis", "Burnetia_mirabilis", "BP_1_7098"))

# Plot taxon groups including convex hulls:
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3, plot_taxon_names = TRUE,
  taxon_groups = taxon_groups, plot_convex_hulls = TRUE)

# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))

# Add root age to tree:
time_tree$root.time <- 269.5

# Prune incomplete taxa from tree:
time_tree <- ape::drop.tip(phy = time_tree, tip = c("Lycaenodon_longiceps",
  "Niuksenitia_sukhonensis"))

# Prune incomplete taxa from cladistic matrix:
cladistic_matrix <- prune_cladistic_matrix(cladistic_matrix = day_2016,
  taxa2prune = c("Lycaenodon_longiceps", "Niuksenitia_sukhonensis"))
```

```
# Note: the above pruning is simply to run this example and should not be
# done manually as a matter of course as the functions will automatically
# prune tips and nodes as required.

# Make new ordination with tree included (enabling phylomorphospace):
pcoa_input <- ordinate_cladistic_matrix(cladistic_matrix = cladistic_matrix,
  time_tree = time_tree)

# Plot this as a simple bivarate phylomorphospace:
plot_morphospace(pcoa_input = pcoa_input)

# Use the Wills technique to add a third axis (PC3):
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3)

# You may need to resize the plot to see the legend for the z-axis

# Add taxon names as well:
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3, plot_taxon_names = TRUE)

# Add taxon groups including convex hulls:
plot_morphospace(pcoa_input = pcoa_input, z_axis = 3, plot_taxon_names = TRUE,
  taxon_groups = taxon_groups, plot_convex_hulls = TRUE)
```

`plot_morphospace_stack` *Plot stacked ordination spaces*

Description

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

Usage

```
plot_morphospace_stack(
  pcoa_input,
  taxon_ages,
  taxon_groups,
  time_bins,
  shear = 0.2,
  x_axis = 1,
  y_axis = 2,
  palette = "viridis",
  plot_cushion = 0.3,
  platform_size = 0.95,
  plot_pillars = TRUE,
  plot_crosshair = TRUE,
  plot_grid_cells = TRUE,
  plot_convex_hulls = TRUE,
```

```
  plot_timebin_names = TRUE,
  plot_tickmarks = TRUE,
  plot_group_legend = TRUE,
  group_legend_position = "bottom_right",
  point_size = 1.5
)
```

Arguments

pcoa_input	The main input in the format output from ordinate_cladistic_matrix.
taxon_ages	A two-column matrix of the first and last apperance dates (columns; "fad" and "lad") for the taxa (as rownames) from pcoa_input.
taxon_groups	A named list of groups to which taxa are assigned (optional). This is used to plot points or convex hulls in different colours corresponding to each group. As the user names the groups these can represent any grouping of interest (e.g., taxonomic, ecological, temporal, spatial).
time_bins	Another two-column matrix of the first and last appearance dates (columns; "fad" and "lad"), this time for named (rownames) time-slices .
shear	A single value (between 0 and 1) that determines the "sheared" visual appearance of the platforms.
x_axis	The ordination axis to plot on the x-axis.
y_axis	The ordination axis to plot nn the y-axis.
palette	The palette to use for plotting each element of taxon_groups. See palette.
plot_cushion	A number determining the "cushion" around the edge of each stack in which no data will be plotted. This should be larger than zero or points will "hang" over the edge. Additionally, if using a platform_size value in excess of one this will avoid points being hidden under overlying platforms. Note that this effectively adds empty plot space around the data, it does not remove anything.
platform_size	The size of each platform as a proportion. Values of less than one will show slight gaps between platforms, whereas values in excess of one will mean platforms will appear to overlap.
plot_pillars	Logical indicating whether or not to plot the pillars linking the corners of each platform.
plot_crosshair	Logical indicating whether or not to plot the "crosshair" (i.e., the zero-zero lines that run through the origin of the morphospace).
plot_grid_cells	Logical indicating whether or not to plot grid cells that help visualise the distorted aspect ratio of the plot. Each cell is a square in the ordination space.
plot_convex_hulls	Logical indicating whether or not to plot convex hulls around the taxonomic groupings. Only relevant if taxon_groups is in use.
plot_timebin_names	Logical indicating whether or not to plot the names of each time bin next to each platform. I.e., the rownames from time_bins. Note if these are long they may disappear behind overlying platforms. To avoid this try using a smaller

platform_size value, a larger shear value, or simply shorter or abbreviated names.

plot_tickmarks Logical indicating whether or not to plot tickmarks next to the bottom platform.

plot_group_legend
Logical indicating whether or not to plot a legend. Only relevant if using taxon_groups. Note this may obscure some points so use with caution and try picking different values for group_legend_position to avoid this.

group_legend_position
The position the group legend should be plotted. Only relevant if using taxon_groups and plot_group_legend = TRUE. Options are: "bottom_left", "bottom_right", "top_left", and "top_right".

point_size The size at which the points should be plotted. Note that here points are custom polygons and hence are not editable by normal plot options, e.g., pch or cex. At present all points are plotted as circles.

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 elsewhere in the literature (e.g., Friedman and Coates 2006; their Figure 2c). Here multiple ordination (or morpho-) spaces are plotted in stratigraphic order (oldest at bottom) as a stacked series of "platforms" representing named time bins.

The user needs to supply three main pieces of information to use the function: 1) ordination data that includes rows (taxa) and columns (ordination axes), 2) the ages (first and last appearance dates) of the taxa sampled, and 3) the ages (first and last appearance dates) of the named time bins used.

Note that since version 0.6.1 this function has been completely rewritten to better reflect the usage of these type of figures (see citations above) as well as allow additional features. This was also done in part to standardise the function to fit the style of the other major disparity plotting functions in Claddis, such as plot_morphospace. This means the input data is now assumed to come directly from ordinate_cladistic_matrix, but the user could easily still bring in data from elsewhere (the way the function worked previously) by reformatting it using something like:

```
pcoa_input <-list(vectors = my_imported_data)
```

Where my_imported_data has columns representing ordination axes (1 to N) and rownames corresponding to taxon names.

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.

See Also

assign_taxa_to_bins, plot_chronophylomorphospace, plot_morphospace, plot_multi_morphospace, ordinate_cladistic_matrix

Examples

```
# Build taxon ages matrix for Day et al 2016 data:
taxon_ages <- matrix(data = c(269, 267, 263, 260, 265, 265, 265, 265, 257, 255, 259, 259, 258, 258,
  260, 257, 257, 255, 257, 257, 255, 252, 259, 259, 260, 258, 253, 252, 257, 255, 257, 255),
  ncol = 2, byrow = TRUE, dimnames = list(c("Biarmosuchus_tener", "Hipposaurus_boonstrai",
  "Bullacephalus_jacksoni", "Pachydectes_elsi", "Lemurosaurus_pricei", "Lobalopex_mordax",
   "Lophorhinus_willodenensis", "Proburnetia_viatkensis", "Lende_chiweta",
  "Paraburnetia_sneeubergensis", "Burnetia_mirabilis", "BP_1_7098", "Niuksenitia_sukhonensis",
   "Ictidorhinus_martinsi", "RC_20", "Herpetoskylax_hopsoni"), c("FAD", "LAD")))

# Ordinate Day et al 2016 data set:
pcoa_input <- ordinate_cladistic_matrix(cladistic_matrix = prune_cladistic_matrix(
  cladistic_matrix = day_2016,
  taxa2prune = "Lycaenodon_longiceps"))

# Build simple taxonomic groups fro Day et al 2016 daat set:
taxon_groups <- list(nonBurnetiamorpha = c("Biarmosuchus_tener", "Hipposaurus_boonstrai",
  "Bullacephalus_jacksoni", "Pachydectes_elsi", "Niuksenitia_sukhonensis", "Ictidorhinus_martinsi",
  "RC_20", "Herpetoskylax_hopsoni"), Burnetiamorpha = c("Lemurosaurus_pricei", "Lobalopex_mordax",
   "Lophorhinus_willodenensis", "Proburnetia_viatkensis", "Lende_chiweta",
   "Paraburnetia_sneeubergensis", "Burnetia_mirabilis", "BP_1_7098"))

# Build a sequence of equally spaced time bins spanning Day et al. 2016 data:
time_sequence <- seq(from = 270, to = 252, length.out = 6)

# Reformt this sequence into named time bin matrix:
time_bins <- matrix(
  data = c(time_sequence[1:(length(x = time_sequence) - 1)],
  time_sequence[2:length(x = time_sequence)]),
  ncol = 2,
  dimnames = list(c("Bin 1", "Bin 2", "Bin 3", "Bin 4", "Bin 5"), c("fad", "lad"))
)

# Plot morphospace stack using named time bins:
plot_morphospace_stack(
  pcoa_input = pcoa_input,
  taxon_ages = taxon_ages,
  taxon_groups = taxon_groups,
  time_bins = time_bins,
```

```
)
```

`plot_multi_morphospace` *Plot Multiple Morphopaces*

Description

Plots multiple morphospaces up to a given number of ordination axes.

Usage

```
plot_multi_morphospace(
  pcoa_input,
  n_axes = 4,
  taxon_groups = NULL,
  plot_taxon_names = FALSE,
  plot_convex_hulls = FALSE,
  plot_internal_nodes = FALSE,
  plot_edges = TRUE,
  plot_root = TRUE,
  root_colour = "red",
  palette = "viridis",
  plot_group_legend = TRUE
)
```

Arguments

`pcoa_input`	The main input in the format outputted from ordinate_cladistic_matrix.
`n_axes`	An integer indicating the total number of axes to plot (should minimally be three).
`taxon_groups`	See plot_morphospace.
`plot_taxon_names`	See plot_morphospace.
`plot_convex_hulls`	See plot_morphospace.
`plot_internal_nodes`	See plot_morphospace.
`plot_edges`	See plot_morphospace.
`plot_root`	See plot_morphospace.
`root_colour`	See plot_morphospace.
`palette`	See plot_morphospace.
`plot_group_legend`	See plot_morphospace.

Details

Takes the output from ordinate_cladistic_matrix and uses plot_morphospace to plot the first N ordination axes.

This allows the user a better appreciation of how variance is distributed across multiple axes and all plots are scaled the saem way to further aid visualisation. Data will seem to "shrink" towards the centre of the space on higher axes as variance decreases.

Most of the options are simply passed to plot_morphospace, but the full range is not available as many will be inappropriate here (e.g., adding a z-axis).

Author(s)

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

See Also

assign_taxa_to_bins, plot_chronophylomorphospace, plot_morphospace_stack, plot_morphospace, ordinate_cladistic_matrix

Examples

```
# Make PCoA for Day 2016 data set:
pcoa_input <- ordinate_cladistic_matrix(cladistic_matrix = day_2016)

# Define some simple taxon groups for the data as a named list:
taxon_groups <- list(nonBurnetiamorpha = c("Biarmosuchus_tener",
   "Hipposaurus_boonstrai", "Bullacephalus_jacksoni", "Pachydectes_elsi",
   "Niuksenitia_sukhonensis", "Ictidorhinus_martinsi", "RC_20",
   "Herpetoskylax_hopsoni", "Lycaenodon_longiceps"),
   Burnetiamorpha = c("Lemurosaurus_pricei", "Lobalopex_mordax",
   "Lophorhinus_willodenensis", "Proburnetia_viatkensis", "Lende_chiweta",
   "Paraburnetia_sneeubergensis", "Burnetia_mirabilis", "BP_1_7098"))

# Plot taxon groups including convex hulls:
plot_multi_morphospace(pcoa_input, n_axes = 5, taxon_groups = taxon_groups,
  plot_convex_hulls = TRUE)

# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))

# Add root age to tree:
```

```
time_tree$root.time <- 269.5

# Make same plot as before but with a phylogeny:
plot_multi_morphospace(
  pcoa_input = pcoa_input,
  n_axes = 5,
  taxon_groups = taxon_groups,
  plot_convex_hulls = TRUE
)
```

`plot_rates_character` *Visualize a rate test time series*

Description

Given the results from a rates test produces a time series visualization for a specific model.

Usage

```
plot_rates_character(test_rates_output, model_number, ...)
```

Arguments

`test_rates_output`	Rate output from test_rates.
`model_number`	The number of the model you wish to visualise from the rate output.
`...`	Other options to be passed to plot.

Details

The raw output from test_rates can be difficult to interpret without visualization and this function provides a means for doing that when the desired output is a time series (other functions will be added for other types of rate test).

The function will only work for a single model, but in practice the user may wish to produce multiple plots in which case they simply need to rn the function multiple times or setup a multipanel window first with layout, or similar.

Plots use the geoscale package to add a geologic time to the x-axis and interested users should consult the documentation there for a full list of options (passed via ...) in the function (see example below).

Calculated rates (changes per lineage million years) are plotted as filled circles and models are plotted as horizontal lines labelled by rate parameters (lambda 1, lmabda 2 etc.).

Value

Nothing is returned, but a plot is produced.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))

# Add root age to tree:
time_tree$root.time <- 269.5

# Prune continuous block from day 2016:
cladistic_matrix <- prune_cladistic_matrix(
  cladistic_matrix = day_2016,
  blocks2prune = 1
)

# Run test rates function for two character partitions:
test_rates_output <- test_rates(
  time_tree = time_tree,
  cladistic_matrix = cladistic_matrix,
  character_partition = list(list(1:34), list(1:17, 18:34)),
  time_bins = seq(from = 270, to = 252, length.out = 10)
)

# Plot 2nd (arbitrary two-partition) character partition model:
plot_rates_character(
  test_rates_output = test_rates_output,
  model_number = 2
)
```

---

plot_rates_time *Visualize a rate test time series*

Description

Given the results from a rates test produces a time series visualization for a specific model.

Usage

```
plot_rates_time(test_rates_output, model_number, ...)
```

Arguments

test_rates_output	Rate output from test_rates.
model_number	The number of the model you wish to visualise from the rate output.
...	Other options to be passed to geoscalePlot.

Details

The raw output from test_rates can be difficult to interpret without visualization and this function provides a means for doing that when the desired output is a time series (other functions will be added for other types of rate test).

The function will only work for a single model, but in practice the user may wish to produce multiple plots in which case they simply need to rn the function multiple times or setup a multipanel window first with layout, or similar.

Plots use the geoscale package to add geologic time to the x-axis and interested users should consult the documentation tere for a full ist of options (passed via ...) in the function (see example below).

Calculated rates (changes per lineage million years) are plotted as filled circles and models are plotted as horizontal lines labelled by rate parameters (lambda_i).

Value

Nothing is returned, but a plot is produced.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))

# Add root age to tree:
time_tree$root.time <- 269.5
```

```
# Prune continuous block from day 2016:
cladistic_matrix <- prune_cladistic_matrix(
  cladistic_matrix = day_2016,
  blocks2prune = 1
)

# Run test rates function for each time bin partition:
test_rates_output <- test_rates(
  time_tree = time_tree,
  cladistic_matrix = cladistic_matrix,
  time_partitions = partition_time_bins(n_time_bins = 9),
  time_bins = seq(from = 270, to = 252, length.out = 10)
)

# Plot 97th time bin partition model:
plot_rates_time(
  test_rates_output = test_rates_output,
  model_number = 97, units = "Stage", cex.ts = 1, cex.age = 1,
  abbrev = "Stage"
)
```

`plot_rates_tree` *Visualize a rate test time series*

Description

Given the results from a rates test produces a time series visualization for a specific model.

Usage

```
plot_rates_tree(test_rates_output, model_type, model_number, ...)
```

Arguments

`test_rates_output`	Rate output from test_rates.
`model_type`	The type of model to plot. Must be one of "branch" or "clade".
`model_number`	The number of the model you wish to visualise from the rate output.
`...`	Other options to be passed to plot.

Details

The raw output from test_rates can be difficult to interpret without visualization and this function provides a means for doing that when the desired output is a time series (other functions will be added for other types of rate test).

The function will only work for a single model, but in practice the user may wish to produce multiple plots in which case they simply need to rn the function multiple times or setup a multipanel window first with layout, or similar.

Plots use the geoscale package to add geologic time to the x-axis and interested users should consult the documentation tere for a full ist of options (passed via ...) in the function (see example below).

Calculated rates (changes per lineage million years) are plotted as filled circles and models are plotted as horizontal lines labelled by rate parameters (lambda_i).

Value

Nothing is returned, but a plot is produced.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Make time-scaled first MPT for Day 2016 data set:
time_tree <- ape::read.tree(text = paste0("(Biarmosuchus_tener:0.5,",
  "(((Hipposaurus_boonstrai:3.5,(Bullacephalus_jacksoni:0.75,",
  "Pachydectes_elsi:0.75):0.75):0.75,(Lemurosaurus_pricei:7.166666667,",
  "(Lobalopex_mordax:4.333333333,((Lophorhinus_willodenensis:3.666666667,",
  "(Proburnetia_viatkensis:0.8333333333,(Lende_chiweta:2,",
  "(Paraburnetia_sneeubergensis:1,Burnetia_mirabilis:2):1):1.833333333)",
  ":0.8333333333):0.8333333333,(BP_1_7098:2.25,Niuksenitia_sukhonensis:",
  "1.25):1.25):0.8333333333):0.8333333333):3.083333333):1.95,",
  "(Ictidorhinus_martinsi:15.9,(RC_20:11.6,(Herpetoskylax_hopsoni:11.3,",
  "Lycaenodon_longiceps:0.3):0.3):0.3):0.3):0.3);"))

# Add root age to tree:
time_tree$root.time <- 269.5

# Prune continuous block from day 2016:
cladistic_matrix <- prune_cladistic_matrix(
  cladistic_matrix = day_2016,
  blocks2prune = 1
)

# Run test rates function for each clade partition:
test_rates_output <- test_rates(
  time_tree = time_tree,
  cladistic_matrix = cladistic_matrix,
  clade_partitions = as.list(x = seq(
    from = ape::Ntip(phy = time_tree) + 1,
    to = ape::Ntip(phy = time_tree) + ape::Nnode(time_tree), by = 1
  )),
  branch_partitions = lapply(X = as.list(x = seq(
    from = 1,
```

```
    to = length(x = time_tree$edge.length), by = 1
  )), as.list),
  time_bins = seq(from = 270, to = 252, length.out = 10)
)

# Plot ninth branch partition model (lowest AIC value):
plot_rates_tree(
  test_rates_output = test_rates_output,
  model_type = "branch", model_number = 9
)

# Plot third clade partition model (lowest AIC value):
plot_rates_tree(
  test_rates_output = test_rates_output,
  model_type = "clade", model_number = 3
)
```

`print.cladisticMatrix` *Compact display of a cladistic matrix*

Description

Displays a compact summary of the dimensions and nature of a cladistic matrix object.

Usage

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

Arguments

`x`	An object of class `"cladisticMatrix"`.
`...`	Further arguments passed to or from other methods.

Details

Displays some basic summary information on a cladistic matrix object, including number and type of characters, information about ordering, and whether variable weights are used.

Value

Nothing is directly returned, instead a text summary describing the dimensions and nature of an object of class `"cladisticMatrix"` is printed to the console.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

build_cladistic_matrix, compactify_matrix, prune_cladistic_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_nexus_matrix, write_tnt_matrix

Examples

```
# Show print.cladisticMatrix version of each included data sets:
print.cladisticMatrix(x = day_2016)
print.cladisticMatrix(x = gauthier_1986)
print.cladisticMatrix(x = michaux_1989)
```

---

prune_cladistic_matrix

Prunes a character matrix of characters or taxa

Description

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

Usage

```
prune_cladistic_matrix(
  cladistic_matrix,
  blocks2prune = c(),
  characters2prune = c(),
  taxa2prune = c(),
  remove_invariant = FALSE
)
```

Arguments

cladistic_matrix	The cladistic matrix in the format imported by read_nexus_matrix.
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(x = cladistic_matrix$matrix).
remove_invariant	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 read_nexus_matrix 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 calculate_morphological_distances) 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

build_cladistic_matrix, compactify_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_nexus_matrix, write_tnt_matrix

Examples

```
# Remove the outgroup taxon and characters 11 and 53 from gauthier_1986:
prunedmatrix <- prune_cladistic_matrix(
  cladistic_matrix =
    gauthier_1986, characters2prune = c(11, 53), taxa2prune =
    c("Outgroup")
)

# Show priuned matrix:
prunedmatrix$matrix_1$matrix
```

---

read_nexus_matrix *Reads in a morphological #NEXUS data file*

Description

Reads in a morphological data file in #NEXUS format.

Usage

```
read_nexus_matrix(file_name, equalize_weights = FALSE)
```

Arguments

file_name	The file name or path of the #NEXUS file.
equalize_weights	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

build_cladistic_matrix, compactify_matrix, prune_cladistic_matrix, safe_taxonomic_reduction, write_nexus_matrix, write_tnt_matrix

Examples

```
# Create example matrix
example_matrix <- paste("#NEXUS", "", "BEGIN DATA;",
                        "\tDIMENSIONS  NTAX=5 NCHAR=5;",
```

```
                        "\tFORMAT SYMBOLS= \" 0 1 2\" MISSING=? GAP=- ;",
                        "MATRIX", "", "Taxon_1  010?0", "Taxon_2  021?0",
                        "Taxon_3  02111", "Taxon_4  011-1",
                        "Taxon_5  001-1", ";", "END;", "",
                        "BEGIN ASSUMPTIONS;",
                        "\tOPTIONS  DEFTYPE=unord PolyTcount=MINSTEPS ;",
                        "\tTYPESET * UNTITLED  = unord: 1 3-5, ord: 2;",
                        "\tWTSET * UNTITLED  = 1: 2, 2: 1 3-5;",
                        "END;", sep = "\n")

# Write example matrix to current working directory called
# "morphmatrix.nex":
cat(example_matrix, file = "morphmatrix.nex")

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

# View example matrix in R:
morph.matrix

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

`safe_taxonomic_reduction` *Safe Taxonomic Reduction*

Description

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

Usage

```
safe_taxonomic_reduction(cladistic_matrix)
```

Arguments

`cladistic_matrix`
: A character-taxon matrix in the format imported by read_nexus_matrix.

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_taxa` 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_taxa`	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.

See Also

build_cladistic_matrix, compactify_matrix, prune_cladistic_matrix, safe_taxonomic_reinsertion, read_nexus_matrix, write_nexus_matrix, write_tnt_matrix

Examples

```
# Performs STR on the Gauthier 1986 dataset used in Wilkinson (1995):
str_data <- safe_taxonomic_reduction(cladistic_matrix = gauthier_1986)

# View deleted taxa:
str_data$str_taxa

# View reduced matrix:
str_data$reduced_matrix

# View removed matrix:
str_data$removed_matrix
```

---

safe_taxonomic_reinsertion

Reinsert Safely Removed Taxa Into A Tree

Description

Safely reinsert taxa in a tree after they were removed from a matrix by Safe Taxonomic Reduction.

Usage

```
safe_taxonomic_reinsertion(
  input_filename,
  output_filename,
  str_taxa,
  multiple_placement_option = "exclude"
)
```

Arguments

input_filename	A Newick-formatted tree file containing tree(s) without safely removed taxa.
output_filename	A file name where the newly generated trees will be written out to (required).
str_taxa	The safe taxonomic reduction table as generated by safe_taxonomic_reduction.
multiple_placement_option	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

The problem with Safe Taxonomic Reduction (safe_taxonomic_reduction) is that it generates trees without the safely removed taxa, but typically the user will ultimately want to include these taxa and thus there is also a need to perform "Safe Taxonomic Reinsertion".

This function performs that task, given a Newick-formatted tree file and a list of the taxa that were safely removed and the senior taxon and rule used to do so (i.e., the $str_taxa part of the output from safe_taxonomic_reduction).

Note that this function operates on tree files rather than reading the trees directly into R (e.g., with ape's read.tree or read.nexus functions) as in practice this turned out to be impractically slow for the types of data sets this function is intended for (supertrees or metatrees). Importantly this means the function operates on raw Newick text strings and hence will only work on data where there is no extraneous information encoded in the Newick string, such as node labels or branch lengths.

Furthermore, in some cases safely removed taxa will have multiple taxa with which they can be safely placed. These come in two forms. Firstly, the multiple taxa can already form a clade, in which case the safely removed taxon will be reinserted in a polytomy with these taxa. In other words, the user should be aware that the function can result in non-bifurcating trees even if the input trees are all fully bifurcating. Secondly, the safely removed taxon can have multiple positions on the tree where it can be safely reinserted. As this generates ambiguity, by default (multiple_placement_option = "exclude") these taxa will simply not be reinserted. However, the user may wish to still incorporate these taxa and so an additional option (multiple_placement_option = "random") allows these taxa to be inserted at any of its' possible positions, chosen at random for each input topology (to give a realistic sense of phylognetic uncertainty. (Note that an exhaustive list of all possible combinations of positions is not implemented as, again, in practice this turned out to generate unfeasibly large numbers of topologies for the types of applications this function is intended for.)

Value

A vector of taxa which were not reinserted is returned (will be empty if all taxa have been reinserted) and a file is written to (`output_filename`).

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

See Also

safe_taxonomic_reduction

Examples

```
# Generate dummy four taxon trees (where taxa B, D and F were
# previously safely excluded):
trees <- ape::read.tree(text = c("(A,(C,(E,G)));", "(A,(E,(C,G)));"))

# Write trees to file:
ape::write.tree(phy = trees, file = "test_in.tre")

# Make dummy safe taxonomic reduction taxon list:
str_taxa <- matrix(data = c("B", "A", "rule_2b", "D", "C", "rule_2b",
  "F", "A", "rule_2b", "F", "C", "rule_2b"), byrow = TRUE, ncol = 3,
  dimnames = list(c(), c("junior", "senior", "rule")))

# Show that taxa B and D have a single possible resinsertion position,
# but that taxon F has two possible positions (with A or with C):
str_taxa

# Resinsert taxa safely (F will be excluded due to the ambiguity of
# its' position - multiple_placement_option = "exclude"):
safe_taxonomic_reinsertion(input_filename = "test_in.tre",
  output_filename = "test_out.tre", str_taxa = str_taxa,
  multiple_placement_option = "exclude")

# Read in trees with F excluded:
exclude_str_trees <- ape::read.tree(file = "test_out.tre")

# Show first tree with B and D reinserted:
ape::plot.phylo(x = exclude_str_trees[[1]])

# Repeat, but now with F also reinserted with its' position (with
# A or with C) chosen at random:
safe_taxonomic_reinsertion(input_filename = "test_in.tre",
  output_filename = "test_out.tre", str_taxa = str_taxa,
  multiple_placement_option = "random")

# Read in trees with F included:
random_str_trees <- ape::read.tree(file = "test_out.tre")
```

```
# Confirm F has now also been reinserted:
ape::plot.phylo(x = random_str_trees[[1]])

# Clean up example files:
file.remove(file1 = "test_in.tre", file2 = "test_out.tre")
```

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

Description

Given a tree and a cladistic-type matrix uses either likelihood ratio tests or the Akaike Information Criterion to compare rate models across branches, clades, time bins, or character partitions.

Usage

```
test_rates(
  time_tree,
  cladistic_matrix,
  time_bins,
  branch_partitions = NULL,
  character_partitions = NULL,
  clade_partitions = NULL,
  time_partitions = NULL,
  change_times = "random",
  test_type = "aic",
  alpha = 0.01,
  multiple_comparison_correction = "benjaminihochberg",
  polymorphism_state = "missing",
  uncertainty_state = "missing",
  inapplicable_state = "missing",
  time_binning_approach = "lloyd",
  all_weights_integers = FALSE,
  estimate_all_nodes = FALSE,
  estimate_tip_values = FALSE,
  inapplicables_as_missing = FALSE,
  polymorphism_behaviour = "equalp",
  uncertainty_behaviour = "equalp",
  threshold = 0.01,
  all_missing_allowed = FALSE
)
```

Arguments

`time_tree`	A tree (phylo object) with branch durations that represents the relationships of the taxa in `cladistic_matrix`.

`cladistic_matrix`
: A character-taxon matrix in the format imported by read_nexus_matrix.

`time_bins`
: A vector of ages (in millions of years) indicating the boundaries of a series of time bins in order from oldest to youngest.

`branch_partitions`
: A list of branch(es) (edge number) partitions to test as N-rate parameter model (where N is the total number of partitions). If NULL (the default) then no partition test(s) will be made.

`character_partitions`
: A list of character partition(s) (character numbers) to test as N-rate parameter model (where N is the total number of partitions). If NULL (the default) then no partition test(s) will be made.

`clade_partitions`
: A list of clade partition(s) (node numbers) to test as N-rate parameter model (where N is the total number of partitions). If NULL (the default) then no partition test(s) will be made.

`time_partitions`
: A list of time bin partition(s) (numbered 1 to N) to test as N-rate parameter model (where N is the total number of partitions). If NULL (the default) then no partition test(s) will be made.

`change_times`
: 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.

`test_type`
: Whether to apply an Akaike Information Criterion (`"aic"`; the default) or likelihood ratio test (`"lrt"`).

`alpha`
: The alpha value to be used for the significance tests. The default is 0.01. Thsi is only relevant if using likelihood ratio tests.

`multiple_comparison_correction`
: Current options are: 1. `"benjaminihochberg"` (the Benjamini and Hochberg 1995 false discovery rate approach; default and recommended), or 2. `"bonferroni"` (the Bonferroni correction). This is only relevant if using likelihood ratio tests.

`polymorphism_state`
: Current options are: 1. `"missing"` (converts polymorphic values to NA; the default), or 2. `"random"` (picks one of the possible polymorphic states at random).

`uncertainty_state`
: Current options are: 1. `"missing"` (converts uncertain values to NA; the default), or 2. `"random"` (picks one of the possible uncertain states at random).

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

`time_binning_approach`
: One of `"close"` or `"lloyd"` (the latter is the default and recommended option).

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

`estimate_all_nodes`
: Option passed to estimate_ancestral_states.

`estimate_tip_values`
: Option passed to estimate_ancestral_states.

`inapplicables_as_missing`
: Option passed to estimate_ancestral_states.

`polymorphism_behaviour`
: Option passed to estimate_ancestral_states.

`uncertainty_behaviour`
: Option passed to estimate_ancestral_states.

`threshold`
: Option passed to estimate_ancestral_states.

`all_missing_allowed`
: Option passed to estimate_ancestral_states.

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 read_nexus_matrix) 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 estimate_ancestral_states function. From here changes along individual branches can be estimated (only the minimum number of changes are inferred; see map_stochastic_changes 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 inferrable rate of evolution) or it will not (the minimum possible inferrable 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) introduced tests by which the significance of an edge (or other partitioning 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:

$$LR = value of likehood function under the null (one-rate) hypothesis / maximum possible value of likehood function u$$

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. Since version 0.4 the option to replace likelihood ratio tests with the Akaike Information Criterion has also been added.

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 `branch_partitions` option).
2. A clade rate (available here with the `clade_partitions` option).
3. A time bin rate (available here with the `time_partitions` option).
4. A character partition rate (available here with the `character_partitions` 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 (based on **ape** numbering, i.e., `plot(tree); edgelabels()`) 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 `branch_partitions = 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, `branch_partitions = lapply(X = as.list(x = 1:nrow(tree$edge)),as.list)`.

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 `branch_partitions = list(list(match(1:ape::Ntip(phy = tree),tree$edge[,2])))`. The `clade_partitions` 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: `clade_partitions = lapply(X = as.list(x = ape::Ntip(phy = tree) + (2:Nnode(tree))),as.list)`. 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). (If using `test_type = "aic"` then the user typically *will* want a value for a single partition.) 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: `time_partitions = lapply(X = as.list(x = 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: `time_partitions = list(list(1:2))`. Or if looking to test a model where each bin has its' own rate value you could use: `time_partitions = list(as.list(x = 1:3))`. Note, as before we do not need to specify the fourth bin as this will be automatically done by the function, however, `time_partitions = list(as.list(x = 1:4))` will also work. Some caution needs to be applied with N-rate models (where N is three or larger) and `test_type = "lrt"` 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 and more genrally if exploring partitions of three or more use of the Akaike Information Criterion (`test_type = "aic"`) is recommended.

Finally, character partitions allow the user to explore whether rates vary across different character types (numbers), e.g., skeletal characters versus soft tissue characters, or cranial characters versus postcranial characters. Here characters are simply numbered 1:N (across all blocks of a matrix), but 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: `character_partitions = list(list(1:10))` to test for a two-rate model with `test_type = "lrt"`.

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 easily realizable 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 up to version 0.2) and unused partition types can be set to NULL, the default in each case.

AIC vs LRT

Since Claddis version 0.4 the option to use the Akaike Information Criterion (AIC) instead of likelihood ratio tests (LRTs) has been added (although it was not properly functional until version 0.4.2). Practically speaking the AIC uses something similar to the denominator term from the LRT (see equation above) and adds a penalty for the number of parameters (partitions). However, it also fundamentally alters the comparative framework applied and hence needs more careful attention from the user to be applied correctly. Specifically, the LRT is by its nature comparative, always comparing an N-rate partition with a one-rate partition. By contrast the AIC does not directly apply any comparison and so the user must logically supply multiple partitionings of the data in order for the results to be meaningful. It might be assumed that a user will always want to apply a single partition that pools all the data for each type of test, whether this is all the edges (branches), time bins, or characters. This will thus be the obvious comparator for any multiple partition supplied, ensuring that any more complex partitioning is minimally superior to this. Additionally, it is also logical to consider each possible way of joining partitions simpler than the most complex partition being considered. E.g., if considering a four-partition model then the user should also consider all possible three-partition and two-partition combinations of that four-partition model. This can obviously lead to some complexity in supplying partitions on the user's part and so some automating of this process is planned in future (but is not fully available yet). For an example, see partition_time_bins.

Additionally, AIC values are not simple to compare as there is no direct equivalent of the alpha value from the LRT. Instead the user can modify the AIC values returned themselves to get delta-AIC or Akaike weight values (e.g., with `geiger::aicw`). (NB: I will not explain these here as there are better explanations online.) Furthermore, since version 0.4.2 sample-size corrected AIC (AICc) is also available in the output. Note that some caution should be used in applying this if the number of partitions is equal to the sample size or only one fewer. E.g., if you have ten time bins and ten partitions you can get a negative value due to the denominator term in the AICc calculation. Thus it is advised to use the raw AIC values as first approximation and be wary if the AICc flips the preferred model to a more complex model or models (i.e., those with more partitions) as this is the opposite of the intent of the AICc.

High versus low rates

Prior to Claddis version 0.3, rate results were classified as significantly high or significntly low as part of the output. This was done simply on whether the estimated p-value fell above or below the corrected alpha level (the significance threshold) and whether the first part of the two-rate partition had a higher or lower rate than the second part (the pooling of all other values). This simple interpretation is no longer valid here as the function can consider more than two partitions (high versus low is meaningless) and the allowing of AIC values means a significance test need not be performed either. Although the same interepretation can still be applied manually when only using two-partition tests and the LRT, other partition sizes and use of the AIC complicate this interpretation (in the same way an ANOVA is more complex than a two-sample t-test). This will also affect visualisation of the data (i.e. the simple pie chart coluring of non-significant, significntly high or significantly low rates seeen since Lloyd et al. 2012 will no longer apply). Instead the user should isolate the best model(s) and attempt to visualise these, perhaps using something like a heat map, with the mean rate for each partition being represented by an appropriate colour.

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 `change_times` 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 potentially make it harder to reject the single-rate null (leading to Type II errors). 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 (i.e., it is only reelevant when `test_type = "lrt"`. 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 (just as 0.05 shouldn't). Note that the function also corrects for multiple comparisons (using the `multiple_comparison_correction` 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 here (albeit not recommended).

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 `polymorphism_state` and `uncertainty_state` (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, especially if the threshold value is set at a high value (see estimate_ancestral_states 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 prune_cladistic_matrix 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 `change_times` option above is used to set the rate numerator, however, completeness remains complex to deal with. The first attempt to deal with this was made by Close et al. (2015) who simply used weighted mean completeness by calculating the proportion of a branch in each bin as the weight and multiplying this by each branch's 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 completeness (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 read_nexus_matrix. However, when weights vary they can create some issues for the function. Specifically, 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 `all_weights_integers` 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 this function.

Finally, the remaining options (`estimate_all_nodes`, `estimate_tip_values`, `inapplicables_as_missing`, `polymorphism_behaviour`, `uncertainty_behaviour`, and `threshold`) are all simply passed directly to estimate_ancestral_states 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.

Output

The output for each LRT test (i.e., the components of the `branch_test_results`, `character_test_results`, `clade_test_results` and `time_test_results` parts of the output) includes three main parts:

1. Rates.
2. p_value.
3. CorrectedAlpha.

Or for each AIC test there are:

1. Rates.
2. AIC.
3. AICc.

For each rate test the `Rates` part of the output is a vector of the absolute rate (number of changes per million years) for each partition in the test (in the order they were supplied to the function). So, for example, a branch rate for the sixth edge in a tree would be the rate for the sixth edge followed by the pooled rate for all other edges. The length of the vector is the length of the number of partitions.

The p_value is a single value indicating the probability that the likelihood ratio (see above and Lloyd et al. 2012) is one, i.e., that the likelihoods of the one-rate and N-rate models are the same.

The CorrectedAlpha is the alpha-value that should be used to determine the significance of the current partition test (i.e., The p_value, above). If the p_value exceeds the CorrectedAlpha then the null (single-rate) hypothesis should be accepted, if lower then the null should be rejected in favour of the N-rate hypthesis. Note that the CorrectedAlpha will not typically be the same for each partition and will also typically be different from the input `alpha` value due to the `multiple_comparison_correction` option used.

The AIC is the Akaike Information Criterion, and is relatively meaningless on its own and can only really be used to compare with the AIC values for other partitions of the data. The AICc is simply the sample-size corrected version of the AIC and is preferable when sample sizes are small.

Value

`time_bins_used` The time binning used (NB: May be slightly altered from the input values).

`inferred_character_changes`
: Matrix of inferred character changes.

`mean_rate` The global (mean) character rate in changes per million years. I.e, the average rate across all characters, branches and time bins, effectively the null hypothesis for any test performed.

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

`branch_test_results`
: List of branch partition results (corresponding to `branch_partitions`. NULL if not requested.

`character_test_results`
: List of character partition results (corresponding to `character_partitions`. NULL if not requested.

`clade_test_results`
: List of clade partition results (corresponding to `clade_partitions`. NULL if not requested.

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

`branch_rates`
: Matrix showing calculated rates for each branch. NULL if `branch_partitions` is not requested.

`character_rates`
: Matrix showing calculated rates for each character. NULL if `character_partitions` is not requested.

`clade_rates`
: Matrix showing calculated rates for each clade. NULL if `clade_partitions` is not requested.

`time_rates`
: Matrix showing calculated rates for each time bin. NULL if `time_partitions` is not requested.

`time_tree`
: The time-scaled input tree used as input (provided as output for use with visualisation functions).

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(seed = 17)

# Generate a random tree for the Michaux data set:
time_tree <- ape::rtree(n = nrow(michaux_1989$matrix_1$matrix))

# Update taxon names to match those in the data matrix:
time_tree$tip.label <- rownames(x = michaux_1989$matrix_1$matrix)

# Set root time by making youngest taxon extant:
time_tree$root.time <- max(diag(x = ape::vcv(phy = time_tree)))

# Get discrete character rates:
x <- test_rates(
  time_tree = time_tree, cladistic_matrix =
    michaux_1989, time_bins = seq(
    from = time_tree$root.time,
    to = 0, length.out = 5
  ), branch_partitions =
    lapply(X = as.list(x = 1:nrow(time_tree$edge)), as.list),
  character_partitions = lapply(
    X =
      as.list(x = 1:3),
    as.list
  ), clade_partitions =
    lapply(
      X =
        as.list(x = ape::Ntip(phy = time_tree) + (2:ape::Nnode(phy = time_tree))),
      as.list
    ), time_partitions =
    lapply(X = as.list(x = 1:4), as.list), change_times =
    "random", alpha = 0.01, polymorphism_state =
    "missing", uncertainty_state = "missing",
  inapplicable_state = "missing", time_binning_approach =
    "lloyd"
)
```

`trim_marginal_whitespace`

Trims marginal whitespace

Description

Trims any marginal whitespace from a vector of character string(s).

Usage

```
trim_marginal_whitespace(x)
```

Arguments

x	A character string

Details

Trims any marginal whitespace (spaces or tabs) from a vector of character string(s).

Value

A vector of character string(s) with any leading or trailing whitespace removed.

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

Examples

```
# Example string:
x <- "   \td s f\t s  "

# Trim only marginal whitespace:
trim_marginal_whitespace(x)
```

---

trim_matrix *Trims a morphological distance matrix*

Description

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

Usage

```
trim_matrix(distance_matrix, tree = NULL)
```

Arguments

distance_matrix	A distance matrix in the format created by calculate_morphological_distances.
tree	If the distance matrix includes ancestors this should be the tree (phylo object) used to estimate their states.

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 estimated ancestral states 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

`distance_matrix`
: 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.

`removed_taxa`
: 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>

See Also

calculate_morphological_distances

Examples

```
# Get morphological distances for Michaux (1989) data set:
distances <- calculate_morphological_distances(cladistic_matrix = michaux_1989)

# Attempt to trim max.distance_matrix:
trim_matrix(distance_matrix = distances$distance_matrix)
```

---

write_nexus_matrix *Writes out a morphological #NEXUS data file*

Description

Writes out a morphological data file in #NEXUS format.

Usage

```
write_nexus_matrix(cladistic_matrix, file_name)
```

Arguments

cladistic_matrix	The cladistic matrix in the format imported by read_nexus_matrix.
file_name	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 read_nexus_matrix.

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

write_tnt_matrix

build_cladistic_matrix, compactify_matrix, prune_cladistic_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_tnt_matrix

Examples

```
# Write out Michaux 1989 to current working directory:
write_nexus_matrix(cladistic_matrix = michaux_1989, file_name = "michaux_1989.nex")

# Remove file when finished:
file.remove(file1 = "michaux_1989.nex")
```

---

write_tnt_matrix *Writes out a morphological TNT data file*

Description

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

Usage

```
write_tnt_matrix(cladistic_matrix, file_name, add_analysis_block = FALSE)
```

Arguments

cladistic_matrix	A cladistic matrix in the format imported by read_nexus_matrix.
file_name	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; Goloboff and Catalano 2016) 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 read_nexus_matrix.

Note that the format can currently deal with continuous characters, sequence (DNA) data, and combinations of these and discrete morphology, but not yet the morphometric format introduced in Goloboff and Catalano (2016).

Author(s)

Graeme T. Lloyd <graemetlloyd@gmail.com>

References

Goloboff, P. A. and Catalano, S. A., 2016. TNT version 1.5, including a full implementation of phylogenetic morphometrics/ *Cladistics*, **32**. 221-238

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

See Also

write_nexus_matrix

build_cladistic_matrix, compactify_matrix, prune_cladistic_matrix, read_nexus_matrix, safe_taxonomic_reduction, write_nexus_matrix

Examples

```
# Write out Michaux 1989 to current working directory:
write_tnt_matrix(cladistic_matrix = michaux_1989, file_name = "michaux_1989.tnt")

# Remove file when finished:
file.remove(file1 = "michaux_1989.tnt")
```

Index