Package 'paleotree'

December 12, 2019

Type Package

Version 3.3.25

Title Paleontological and Phylogenetic Analyses of Evolution

Date 2019-12-12

Author David W. Bapst, Peter J. Wagner

Depends R (>= 3.0.0), ape (>= 4.1)

Imports phangorn (>= 2.0.0), stats, phytools (>= 0.6-00), jsonlite, graphics, grDevices, methods, png, RCurl, utils

Suggests spelling, testthat

Language en-US

ByteCompile TRUE

Encoding UTF-8

Maintainer David W. Bapst <dwbapst@gmail.com>

BugReports https://github.com/dwbapst/paleotree/issues

Description Provides tools for transforming, a posteriori time-scaling, and modifying phylogenies containing extinct (i.e. fossil) lineages. In particular, most users are interested in the functions timePaleoPhy, bin_timePaleoPhy, cal3TimePaleoPhy and bin_cal3TimePaleoPhy, which date cladograms of fossil taxa using stratigraphic data. This package also contains a large number of likelihood functions for estimating sampling and diversification rates from different types of data available from the fossil record (e.g. range data, occurrence data, etc). paleotree users can also simulate diversification and sampling in the fossil record using the function simFossilRecord, which is a detailed simulator for branching birth-death-sampling processes composed of discrete taxonomic units arranged in ancestor-descendant relationships. Users can use simFossilRecord to simulate diversification in incompletely sampled fossil records, under various models of morphological differentiation (i.e. the various patterns by which morphotaxa originate from one another), and with time-dependent, longevity-dependent and/or diversity-dependent rates of diversification, extinction and sampling. Additional functions allow users to translate simulated ancestor-descendant data from simFossilRecord into standard time-scaled phylogenies or unscaled cladograms that reflect the relationships among taxon units.

License CC0

URL https://github.com/dwbapst/paleotree

RoxygenNote 7.0.2

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-12 12:50:02 UTC

R topics documented:

paleotree-package *paleotree: Paleontological and Phylogenetic Analyses of Evolution*

Description

Analyzes, time-scales and simulates phylogenies of extinct/fossil lineages, along with calculation of diversity curves. Also fits likelihood models to estimate sampling rates from stratigraphic ranges.

Details

Package:	paleotree
Type:	Package
License:	CC0

This package contains functions for analyzing sampling rates given ranges of fossil taxa, in both continuous and discrete time, functions for *a posteriori* time-scaling phylogenies of fossil taxa and functions for simulating the fossil record in both taxic and phylogenetic varieties.

Author(s)

David W. Bapst

Maintainer: David W. Bapst <dwbapst@gmail.com>

References

Bapst, D.W. 2012. paleotree: an R package for paleontological and phylogenetic analyses of evolution. *Methods in Ecology and Evolution*. 3: 803-807. doi: 10.1111/j.2041-210X.2012.00223.x

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Bapst, D. W. 2013. When Can Clades Be Potentially Resolved with Morphology? *PLoS ONE*. 8(4):e62312.

Bapst, D. W. 2014. Assessing the effect of time-scaling methods on phylogeny-based analyses in the fossil record. **Paleobiology 40**(3):331-351.

See Also

This package relies extensively on the phylogenetic toolkit and standards offered by the ape package, and hence lists this package as a depends, so it is loaded simultaneously.

Examples

```
# get the package version of paleotree
packageVersion("paleotree")
```

```
# get the citation for paleotree
citation("paleotree")

## Simulate some fossil ranges with simFossilRecord
set.seed(444);
record <- simFossilRecord(
     p = 0.1, q = 0.1,
     nruns = 1,
     nTotalTaxa = c(30,40),
     nExtant = 0
     )
taxa <- fossilRecord2fossilTaxa(record)

# let's see what the 'true' diversity curve looks like in this case
   # plot the FADs and LADs with taxicDivCont()
taxicDivCont(taxa)

# simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.5)

# plot the diversity curve based on the sampled ranges
layout(1:2)
taxicDivCont(rangesCont)

# Now let's use binTimeData to bin in intervals of 10 time units
rangesDisc <- binTimeData(rangesCont,int.length = 10)

# plot with taxicDivDisc
taxicDivDisc(rangesDisc)

#compare to the continuous time diversity curve above!

layout(1)

# taxa2phylo assumes we know speciation events perfectly... what if we don't?

# first, let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot = TRUE)

# Now let's try timePaleoPhy using the continuous range data
ttree <- timePaleoPhy(cladogram,rangesCont,type = "basic",plot = TRUE)

# plot diversity curve
phyloDiv(ttree,drop.ZLB = TRUE)

# that tree lacked the terminal parts of ranges (tips stops at the taxon FADs)
# let's add those terminal ranges back on with add.term
ttree <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    add.term = TRUE,
    plot = TRUE
```

```
    )

# plot diversity curve
phyloDiv(ttree)
```

`binTimeData` *Bin Simulated Temporal Ranges in Discrete Intervals*

Description

Converts a matrix of simulated continuous-time first occurrences and last occurrences for fossil taxa into first and last occurrences given in some set of discrete-time intervals, either simulated or place *a priori*, which is output along with information of the dates of the given intervals.

Usage

```
binTimeData(timeData, int.length = 1, start = NA, int.times = NULL)
```

Arguments

`timeData`	Two-column matrix of simulated first and last occurrences in absolute continuous time.
`int.length`	Time interval length, default is 1 time-unit.
`start`	Starting time for calculating the intervals.
`int.times`	A two column matrix with the start and end times of the intervals to be used.

Details

This function takes a simulated matrix of per-taxon first and last occurrences and, by dividing the time-scale into time intervals of non-zero length, lists taxon occurrences within those interval. By default, a set of sequential non-overlapping time-interval of equal non-zero length are used, with the length controlled by the argument int.length.

Alternatively, a two column matrix of interval start and end times to be used can be input via the argument `int.times`. None of these intervals can have a duration (temporal length) greater than zero. If a first or last appearance in the input range data could fit into multiple intervals (i.e. the input discrete time intervals are overlapping), then the appearance data is placed in the interval of the shortest duration. When output, the interval times matrix (see below) will be sorted from first to last.

As with many functions in the `paleotree` package, absolute time is always decreasing, i.e. the present day is zero. However, the numbering of intervals giving in the output increases with time, as these are numbered relative to each other, from first to last.

As of version 1.7 of `paleotree`, taxa which are extant as indicated in `timeData` as being in a time interval bounded `(0,0)`, unless time-bins are preset using argument `int.times` (prior to version 1.5 they were erroneously listed as NA).

Value

A list containing:

`int.times`	A 2-column matrix with the start and end times of the intervals used; time decreases relative to the present.
`taxon.times`	A 2-column matrix with the first and last occurrences of taxa in the intervals listed in `int.times`, with numbers referring to the row of `int.times`.

Note

This function is *SPECIFICALLY* for simulating the effect of having a discrete time-scale for analyses using simulations. This function should not be used for non-simulations uses, such as binning temporal occurrences for analyses of real data. In those case, the temporal ranges (which, in real data, will probably be given as discrete time intervals) should already be tabulated within discrete intervals prior to use in `paleotree`. The user should place the temporal information in a `list` object, as described for the output of `binTimeData`.

Author(s)

David W. Bapst

See Also

`simFossilRecord`, `sampleRanges`, `taxicDivCont`

Examples

```
# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          nruns = 1,
                          nTotalTaxa = c(30,40),
                          nExtant = 0
                          )
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record with imperfect sampling via sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.5)
# Now let's use binTimeData() to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length = 1)
# plot with taxicDivDisc()
equalDiscInt <- taxicDivDisc(rangesDisc)

# example with pre-set intervals input (including overlapping)
presetIntervals <- cbind(
    c(1000, 990, 970, 940),
    c(980, 970, 950, 930)
    )
rangesDisc1 <- binTimeData(rangesCont,
    int.times = presetIntervals)
```

```
# plot the diversity curve with these uneven bins
taxicDivDisc(rangesDisc1)

# now let's plot the diversity from these unequal-length bins
   # with the original equal length intervals from above
taxicDivDisc(rangesDisc1, int.times = equalDiscInt[,1:2])


####################################
#example with extant taxa
set.seed(444)
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          nruns = 1,
                          nTotalTaxa = c(30,40)
                          )
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record
    # with imperfect sampling via sampleRanges
rangesCont <- sampleRanges(
    taxa, r = 0.5,
    modern.samp.prob = 1)
# Now let's use binTimeDat to bin into intervals of 1 time-unit
rangesDisc <- binTimeData(rangesCont,
    int.length = 1)
# plot with taxicDivDisc()
taxicDivDisc(rangesDisc)

# example with pre-set intervals input
    # (including overlapping)
presetIntervals <- cbind(
    c(40, 30, 20, 10),
    c(30, 20, 10, 0)
    )
rangesDisc1 <- binTimeData(rangesCont,
    int.times = presetIntervals)

taxicDivDisc(rangesDisc1)
```

`branchClasses` *Partitions the branch lengths of a tree into several classes based on their placement.*

Description

Partitions the branch lengths of a tree into several classes based on their placement.

Usage

```
branchClasses(tree, whichExtant = NULL, tol = 0.01)
```

Arguments

tree	A dated phylogeny to be analyzed, as an object of class phylo, ideally with a $root.time element as is typical for paleotree output phylogenies. If $root.time is not present, the most recent tips will be interpreted as being at the modern day (i.e. 0 time-units before present).
whichExtant	A logical vector with length equal to number of tips in the tree. A TRUE value indicates that the respective tip taxon (as indicated by the ordering of the tip labels) that is extant at the modern day, while FALSE values equate to the respect being extinct at the present day. If present, this vector is used for determining which taxa are extant, and which are extinct.
tol	Tolerance used to distinguish extant taxa, if whichExtant is not provided, to avoid issues with number rounding. Taxa within tol of the modern day will be considered extant.

Details

This function will partition the internode (node to node, including internal node to terminal tip) branch lengths of a tree into four separate classes: all (all the internode branches of a tree), int (internal branches which run from one internode to another), live (terminal branches which run from an internal node to a terminal tip representing an extinction event before the present) and dead (terminal branches which run from an internal node to a terminal tip at the modern day, reflecting a still-living taxon).

The depths of the internal 'mother' node (i.e. time of origin, before the modern day) of each branch length are included as the labels of the branch length vectors.

This function is mainly of use for modeling internode branch lengths in a phylogeny including fossil taxa.

Value

The output is a list consisting of four vectors, with the labels of the vectors being their corresponding time of origin. See details.

Examples

```
#simulated example
set.seed(444)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = c(10,20)
    )
taxa <- fossilRecord2fossilTaxa(record)
```

```
tree <- taxa2phylo(taxa)
brlenRes <- branchClasses(tree)

#see frequency histograms of branch lengths
layout(1:4)
for(x in 1:length(brlenRes)){
hist(
    brlenRes[[x]],
    main = "Branch Lengths",
    xlab = names(brlenRes)[x])
}

#see frequency histograms of branch depths
layout(1:4)
for(x in 1:length(brlenRes)){
hist(
    as.numeric(names(brlenRes[[x]])),
    main = "Branch Depths",
    xlab = names(brlenRes)[x])
}

layout(1)
```

`cal3TimePaleoPhy` *Three Rate Calibrated* a posteriori *Dating of Paleontological Phylogenies*

Description

Time-scales an undated cladogram of fossil taxa, using information on their ranges and estimates of the instantaneous rates of branching, extinction and sampling. The output is a sample of *a posteriori* time-scaled trees, as resulting from a stochastic algorithm which samples observed gaps in the fossil record with weights calculated based on the input rate estimates. This function also uses the three-rate calibrated dating algorithm to stochastically resolve polytomies and infer potential ancestor-descendant relationships, simultaneous with the time-scaling treatment.

Usage

```
cal3TimePaleoPhy(
  tree,
  timeData,
  brRate,
  extRate,
  sampRate,
  ntrees = 1,
  anc.wt = 1,
  node.mins = NULL,
  dateTreatment = "firstLast",
  FAD.only = FALSE,
```

```
  adj.obs.wt = TRUE,
  root.max = 200,
  step.size = 0.1,
  randres = FALSE,
  noisyDrop = TRUE,
  verboseWarnings = TRUE,
  diagnosticMode = FALSE,
  tolerance = 1e-04,
  plot = FALSE
)

bin_cal3TimePaleoPhy(
  tree,
  timeList,
  brRate,
  extRate,
  sampRate,
  ntrees = 1,
  anc.wt = 1,
  node.mins = NULL,
  dateTreatment = "firstLast",
  FAD.only = FALSE,
  sites = NULL,
  point.occur = FALSE,
  nonstoch.bin = FALSE,
  adj.obs.wt = TRUE,
  root.max = 200,
  step.size = 0.1,
  randres = FALSE,
  noisyDrop = TRUE,
  verboseWarnings = TRUE,
  tolerance = 1e-04,
  diagnosticMode = FALSE,
  plot = FALSE
)
```

Arguments

tree	An unscaled cladogram of fossil taxa, of class phylo. Tip labels must match the taxon labels in the respective temporal data.
timeData	Two-column matrix of first and last occurrences in absolute continuous time, with row names as the taxon IDs used on the tree. This means the first column is very precise FADs (first appearance dates) and the second column is very precise LADs (last appearance dates), reflect the precise points in time when taxa first and last appear. If there is stratigraphic uncertainty in when taxa appear in the fossil record, it is preferable to use the bin_ dating functions; however, see the argument dateTreatment.
brRate	Either a single estimate of the instantaneous rate of branching (also known as

the 'per-capita' origination rate, or speciation rate if taxonomic level of interest is species) or a vector of per-taxon estimates

`extRate` Either a single estimate of the instantaneous extinction rate (also known as the 'per-capita' extinction rate) or a vector of per-taxon estimates

`sampRate` Either a single estimate of the instantaneous sampling rate or a vector of per-taxon estimates

`ntrees` Number of dated trees to output.

`anc.wt` Weighting against inferring ancestor-descendant relationships. The argument `anc.wt` allows users to alter the default consideration of ancestor-descendant relationships. This value is used as a multiplier applied to the probability of choosing any node position which would infer an ancestor-descendant relationship. By default, `anc.wt = 1`, and thus these probabilities are unaltered. if `anc.wt` is less than 1, the probabilities decrease and at `anc.wt = 0`, no ancestor-descendant relationships are inferred at all. Can be a single value or a vector of per-taxon values, such as if a user wants to only allow plesiomorphic taxa to be ancestors.

`node.mins` The minimum dates of internal nodes (clades) on a phylogeny can be set using `node.mins`. This argument takes a vector of the same length as the number of nodes, with dates given in the same order as nodes are ordered in the `tree$edge` matrix. Note that in `tree$edge`, terminal tips are given the first set of numbers (`1:Ntip(tree)`), so the first element of `node.mins` is the first internal node (the node numbered `Ntip(tree)+1`, which is generally the root for most `phylo` objects read by `read.tree`). Not all nodes need be given minimum dates. Nodes without minimum dates can be given as NA in `node.mins`, but the vector must be the same length as the number of internal nodes in `tree`. These are minimum date constraints, such that a node will be 'frozen' by the cal3 algorithm so that constrained nodes will always be *at least as old as this date*, but the final date may be even older depending on the taxon dates used, the parameters input for the cal3 algorithm and any other minimum node dates given (e.g. if a clade is given a very old minimum date, this will (of course) over-ride any minimum dates given for clades that that node is nested within). if the constrained nodes include a polytomy, this polytomy will still be resolved with respect to the cal3 algorithm, but the first divergence will be 'frozen' so that it is at least as old as the minimum age, while any additional divergences will be allowed to occur after this minimum age.

`dateTreatment` This argument controls the interpretation of `timeData`. The default setting `dateTreatment = "firstLast"` treats the dates in `timeData` as a column of precise first and last appearances.

A second option is `dateTreatment = "minMax"`, which treats these dates as minimum and maximum bounds on single point dates. Under this option, all taxa in the analysis will be treated as being point dates, such that the first appearance is also the last. These dates will be pulled under a uniform distribution. Note that use of `dateTreatment = "minMax"` was bugged in versions paleotree <3.2.5, as the generating time-tree used for cal3 inference was generated using the input `timeData` as fixed first and last dates, such that the effect of `dateTreatment = "minMax"` was nearly identical to using `dateTreatment = "firstLast"` regardless of the arguments chosen by a user, with tips always being placed at the upper date constraint as if the time of observation was a fixed

LAD. This is now fixed as v3.2.6, such that a new basic-time-scaled tree is generated from a randomly selected set of point occurrence times on each iteration, so that the resulting tip dates are always different.

A third option is `dateTreatment = "randObs"`, which assumes that the dates in the matrix are first and last appearance times, but that the desired time of observation is unknown. Thus, this is much like `dateTreatment = "firstLast"` except the effective time of observation (the taxon's LAD under `dateTreatment = "firstLast"`) is treated as an uncertain date, and is randomly sampled between the first and last appearance times. The FAD still is treated as a fixed number, used for dating the nodes. In previous versions of paleotree, this was called in `cal3timePaleoPhy` using the argument `rand.obs`, which has been removed for clarity. This temporal uncertainty in times of observation might be useful if a user is interested in applying phylogeny-based approaches to studying trait evolution, but have per-taxon measurements of traits that come from museum specimens with uncertain temporal placement.

With both arguments `dateTreatment = "minMax"` and `dateTreatment = "randObs"`, the time of observation of taxa is a point-occurrence with a free-floating random variable as the precise age. Thus, the option `FAD.only = TRUE` is incoherent with these other options for `dateTreatment`, and thus their use together will return an error message. Furthermore, the sampling of dates from random distributions in these approaches should compel users to produce many time-scaled trees for any given analytical purpose.

Note that `dateTreatment = "minMax"` returns an error in '`bin_`' time-scaling functions; please use `points.occur` instead.

`FAD.only` Should the tips represent observation times at the start of the taxon ranges? `FAD.only = TRUE`, the resulting output is similar to when terminal ranges are no added on with `timePaleoPhy`. If `FAD.only = TRUE` and `dateTreatment = "minMax"` or `dateTreatment = "randObs"`, the function will stop and a warning will be produced, as these combinations imply contradictory sets of times of observation.

`adj.obs.wt` If the time of observation of a taxon is before the last appearance of that taxon, should the weight of the time of observation be adjusted to account for the known observed history of the taxon which occurs *after* the time of observation? If so, then set `adj.obs.wt = TRUE`. This argument should only have an effect if time of observation *IS NOT* the LAD, if the times of observation for for a potential ancestor are earlier than the first appearance of their potential descendants, *and* if the ancestral weights for taxa are not set to zero (so there can be potential ancestors).

`root.max` Maximum time before the first FAD that the root can be pushed back to.

`step.size` Step size of increments used in zipper algorithm to assign node ages.

`randres` Should polytomies be randomly resolved using `multi2di` in ape rather than using the cal3 algorithm to weight the resolution of polytomies relative to sampling in the fossil record?

`noisyDrop` If `TRUE` (the default), any taxa dropped from tree due to not having a matching entry in the time data will be listed in a system message.

`verboseWarnings`
: if `TRUE` (the default), then various warnings and messages regarding best practices will be issued to the console about the analysis. If `FALSE`,the function will run as quietly as possible.

`diagnosticMode`
: If `TRUE`, `cal3timePaleoPhy` will return to the console and to graphic devices an enormous number of messages, plots and ancillary information that may be useful or entirely useless to figuring out what is going wrong.

`tolerance`
: Acceptable amount of shift in tip dates from dates listed in `timeData`. Shifts outside of the range of `tolerance` will cause a warning message to be issued that terminal tips appear to be improperly aligned.

`plot`
: If true, plots the input, "basic" time-scaled phylogeny (an intermediary step in the algorithm) and the output cal3 time-scaled phylogeny.

`timeList`
: A list composed of two matrices giving interval times and taxon appearance dates. The rownames of the second matrix should be the taxon IDs, identical to the `tip.labels` for tree. See details.

`sites`
: Optional two column matrix, composed of site IDs for taxon FADs and LADs. The sites argument allows users to constrain the placement of dates by restricting multiple fossil taxa whose FADs or LADs are from the same very temporally restricted sites (such as fossil-rich Lagerstatten) to always have the same date, across many iterations of time-scaled trees. To do this, provide a `matrix` to the `sites` argument where the "site" of each FAD and LAD for every taxon is listed, as corresponding to the second matrix in `timeList`. If no sites matrix is given (the default), then it is assumed all fossil come from different "sites" and there is no shared temporal structure among the events.

`point.occur`
: If true, will automatically produce a 'sites' matrix which forces all FADs and LADs to equal each other. This should be used when all taxa are only known from single 'point occurrences', i.e. each is only recovered from a single bed/horizon, such as a Lagerstatten.

`nonstoch.bin`
: If `nonstoch.bin = TRUE` (the default is `FALSE`, dates are *not* stochastically drawn from uniform distributions bounded by the upper and lower boundaries of the geologic intervals (the 'bins'), as typically occurs with '`bin_`' time-scaling methods in `paleotree` but instead first-appearance dates are assigned to the earliest time of the interval a taxon first appears in, while last-appearance dates are placed at the youngest (the 'later-most') date in the interval that that taxon last appears in. This option may be useful for plotting. Note that if `nonstoch.bin = TRUE`, the `sites` argument becomes arbitrary and has no influence on the output.

Details

The three-rate calibrated ("cal3") algorithm time-scales trees *a posteriori* by stochastically picking node divergence times relative to a probability distribution of expected waiting times between speciation and first appearance in the fossil record. This algorithm is extended to apply to resolving polytomies and designating possible ancestor-descendant relationships. The full details of this method are provided in Bapst (2013, MEE).

Briefly, cal3 time-scaling is done by examining each node separately, moving from the root upwards. Ages of branching nodes are constrained below by the ages of the nodes below them (except the root; hence the need for the root.max argument) and constrained above by the first appearance

dates (FADs) of the daughter lineages. The position of the branching event within this constrained range implies different amounts of unobserved evolutionary history. cal3 considers a large number of potential positions for the branching node (the notation in the code uses the analogy of viewing the branching event as a 'zipper') and calculates the summed unobserved evolutionary history implied by each branching time. The probability density of each position is then calculated under a gamma distribution with a shape parameter of 2 (implying that it is roughly the sum of two normal waiting times under an exponential) and a rate parameter which takes into account both the probability of not observing a lineage of a certain duration and the 'twiginess' of the branch, i.e. the probability of having short-lived descendants which went extinct and never were sampled (similar to Friedman and Brazeau, 2011). These densities calculated under the gamma distribution are then used as weights to stochastically sample the possible positions for the branching node. This basic framework is extended to polytomies by allowing a branching event to fall across multiple potential lineages, adding each lineage one by one, from earliest appearing to latest appearing (the code notation refers to this as a 'parallel zipper').

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

These functions will intuitively drop taxa from the tree with NA for range or that are missing from `timeData`.

The sampling rate used by cal3 methods is the instantaneous sampling rate, as estimated by various other function in the paleotree package. See `make_durationFreqCont` for more details. If you have the per-time unit sampling probability ('R' as opposed to 'r') look at the sampling parameter conversion functions also included in this package (e.g. `sProb2sRate`). Most datasets will probably use `make_durationFreqDisc` and `sProb2sRate` prior to using this function, as shown in an example below.

The branching and extinction rate are the 'per-capita' instantaneous origination/extinction rates for the taxic level of the tips of the tree being time-scaled. Any user of the cal3 time-scaling method has multiple options for estimating these rates. One is to separately calculate the per-capita rates (following the equations in Foote, 2001) across multiple intervals and take the mean for each rate. A second, less preferred option, would be to use the extinction rate calculated from the sampling rate above (under ideal conditions, this should be very close to the mean 'per-capita' rate calculated from by-interval FADs and LADs). The branching rate in this case could be assumed to be very close to the extinction rate, given the tight relationship observed in general between these two (Stanley, 1976; see Foote et al., 1999, for a defense of this approach), and thus the extinction rate estimate could be used also for the branching rate estimate. (This is what is done for the examples below.) A third option for calculating all three rates simultaneously would be to apply likelihood methods developed by Foote (2002) to forward and reverse survivorship curves. Note that only one of these three suggested methods is implemented in `paleotree`: estimating the sampling and extinction rates from the distribution of taxon durations via `make_durationFreqCont` and `make_durationFreqDisc`.

By default, the cal3 functions will consider that ancestor-descendant relationships may exist among the given taxa, under a budding cladogenetic or anagenetic modes. Which tips are designated as which is given by two additional elements added to the output tree, `$budd.tips` (taxa designated as ancestors via budding cladogenesis) and `$anag.tips` (taxa designated as ancestors via anagenesis). This can be turned off by setting `anc.wt = 0`. As this function may infer anagenetic relationships during time-scaling, this can create zero-length terminal branches in the output. Use `dropZLB` to get rid of these before doing analyses of lineage diversification.

Unlike `timePaleoPhy`, cal3 methods will always resolve polytomies. In general, this is done using

the rate calibrated algorithm, although if argument randres = TRUE, polytomies will be randomly resolved with uniform probability, similar to multi2di from ape. Also, cal3 will always add the terminal ranges of taxa. However, because of the ability to infer potential ancestor-descendant relationships, the length of terminal branches may be shorter than taxon ranges themselves, as budding may have occurred during the range of a morphologically static taxon. By resolving polytomies with the cal3 method, this function allows for taxa to be ancestral to more than one descendant taxon. Thus, users who believe their dataset may contain indirect ancestors are encouraged by the package author to try cal3 methods with their consensus trees, as opposed to using the set of most parsimonious trees. Comparing the results of these two approaches may be very revealing.

Like timePaleoPhy, cal3TimePaleoPhy is designed for direct application to datasets where taxon first and last appearances are precisely known in continuous time, with no stratigraphic uncertainty. This is an uncommon form of data to have from the fossil record, although not an impossible form (micropaleontologists often have very precise range charts, for example). This means that most users *should not* use cal3TimePaleoPhy directly, unless they have written their own code to deal with stratigraphic uncertainty. For some groups, the more typical 'first' and 'last' dates represent the minimum and maximum absolute ages for the fossil collections that a taxon is known is known from. Presumably, the first and last appearances of that taxon in the fossil record is at unknown dates within these bounds. These should not be mistaken as the FADs and LADs desired by cal3TimePaleoPhy, as cal3TimePaleoPhy will use the earliest dates provided to calibrate node ages, which is either an overly conservative approach to time-scaling or fairly nonsensical.

If you have time-data in discrete intervals, consider using bin_cal3TimePaleoPhy as an alternative to cal3TimePaleoPhy.

bin_cal3TimePaleoPhy is a wrapper of cal3TimePaleoPhy which produces time-scaled trees for datasets which only have interval data available. For each output tree, taxon first and last appearance dates are placed within their listed intervals under a uniform distribution. Thus, a large sample of time-scaled trees will approximate the uncertainty in the actual timing of the FADs and LADs.

The input timeList object can have overlapping (i.e. non-sequential) intervals, and intervals of uneven size. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. If taxa occur only in single collections (i.e. their first and last appearance in the fossil record is synchronous, the argument point.occur will force all taxa to have instantaneous durations in the fossil record. Otherwise, by default, taxa are assumed to first and last appear in the fossil record at different points in time, with some positive duration. The sites matrix can be used to force only a portion of taxa to have simultaneous first and last appearances.

If timeData or the elements of timeList are actually data frames (as output by read.csv or read.table), these will be coerced to a matrix.

A tutorial for applying the time-scaling functions in paleotree, particularly the cal3 method, along with an example using real (graptolite) data, can be found at the following link:

http://nemagraptus.blogspot.com/2013/06/a-tutorial-to-cal3-time-scaling-using.html

Value

The output of these functions is a time-scaled tree or set of time-scaled trees, of either class phylo or multiPhylo, depending on the argument ntrees. All trees are output with an element $root.time. This is the time of the root on the tree and is important for comparing patterns across trees.

Additional elements are `sampledLogLike` and `$sumLogLike` which respectively record a vector containing the 'log-densities' of the various node-ages selected for each tree by the 'zipper' algorithm, and the sum of those log-densities. Although they are very similar to log-likelihood values, they are not true likelihoods, as node ages are conditional on the other ages selected by other nodes. However, these values may give an indication about the relative optimality of a set of trees output by the cal3 functions.

Trees created with `bin_cal3TimePaleoPhy` will output with some additional elements, in particular `$ranges.used`, a matrix which records the continuous-time ranges generated for time-scaling each tree (essentially a pseudo-`timeData` matrix.)

Note

Most importantly, please note the stochastic element of the three rate-calibrated time-scaling methods. These do not use traditional optimization methods, but instead draw divergence times from a distribution defined by the probability of intervals of unobserved evolutionary history. This means analyses MUST be done over many cal3 time-scaled trees for analytical rigor! No one tree is correct.

Similarly, please account for stratigraphic uncertainty in your analysis. Unless you have exceptionally resolved data, use a wrapper with the cal3 function, either the provided `bin_cal3TimePaleoPhy` or code a wrapper function of your own that accounts for stratigraphic uncertainty in your dataset. Remember that the FADs (earliest dates) given to `timePaleoPhy` will *always* be used to calibrate node ages!

Author(s)

David W. Bapst

References

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Foote, M. 2000. Origination and extinction components of taxonomic diversity: general problems. Pp. 74-102. In D. H. Erwin, and S. L. Wing, eds. *Deep Time: Paleobiology's Perspective.* The Paleontological Society, Lawrence, Kansas.

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Friedman, M., and M. D. Brazeau. 2011. Sequences, stratigraphy and scenarios: what can we say about the fossil record of the earliest tetrapods? *Proceedings of the Royal Society B: Biological Sciences* 278(1704):432-439.

Stanley, S. M. 1979. Macroevolution: Patterns and Process. W. H. Freeman, Co., San Francisco.

See Also

`timePaleoPhy`, `make_durationFreqCont`, `pqr2Ps`, `sProb2sRate`, `multi2di`

Examples

```
# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          nruns = 1,
                         nTotalTaxa = c(30,40),
                          nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)

# simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,
      r = 0.5)
# let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,
      plot = TRUE)

# this package allows one to use
  # rate calibrated type time-scaling methods (Bapst, 2014)
# to use these, we need an estimate of the sampling rate
     # (we set it to 0.5 above)
likFun <- make_durationFreqCont(rangesCont)
srRes <- optim(
    parInit(likFun),
    likFun,
    lower = parLower(likFun),
    upper = parUpper(likFun),
    method = "L-BFGS-B",
    control = list(maxit = 1000000))
sRate <- srRes[[1]][2]

# we also need extinction rate and branching rate
   # we can get extRate from getSampRateCont too
# we'll assume extRate = brRate (ala Foote et al., 1999)
    # this may not always be a good assumption!
divRate <- srRes[[1]][1]

# now let's try cal3TimePaleoPhy
   # which time-scales using a sampling rate to calibrate
# This can also resolve polytomies based on
    # sampling rates, with some stochastic decisions
ttree <- cal3TimePaleoPhy(
    cladogram,
     rangesCont,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate,
    ntrees = 1,
    plot = TRUE)

# notice the warning it gives!
```

```
phyloDiv(ttree)

# by default, cal3TimePaleoPhy may predict indirect ancestor-descendant relationships
     # can turn this off by setting anc.wt = 0
ttree <- cal3TimePaleoPhy(
    cladogram,
     rangesCont,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate,
    ntrees = 1,
    anc.wt = 0,
    plot = TRUE)


# let's look at how three trees generated
    # with very different time of obs. look

ttreeFAD <- cal3TimePaleoPhy(
    cladogram,
    rangesCont,
    brRate = divRate,
    extRate = divRate,
    FAD.only = TRUE,
    dateTreatment = "firstLast",
    sampRate = sRate,
    ntrees = 1,
    plot = TRUE)

ttreeRand <- cal3TimePaleoPhy(
    cladogram,
    rangesCont,
    brRate = divRate,
    extRate = divRate,
    FAD.only = FALSE,
    dateTreatment = "randObs",
    sampRate = sRate,
    ntrees = 1,plot = TRUE)

# by default the time of observations are the LADs
ttreeLAD <- cal3TimePaleoPhy(
    cladogram,
    rangesCont,
    brRate = divRate,
    extRate = divRate,
    FAD.only = FALSE,
    dateTreatment = "randObs",
    sampRate = sRate,
    ntrees = 1,
    plot = TRUE)

# and let's plot
layout(1:3)
```

```
parOrig <- par(no.readonly = TRUE)
par(mar = c(0,0,0,0))
plot(ladderize(ttreeFAD));text(5,5,
    "time.obs = FAD",
    cex = 1.5, pos = 4)
plot(ladderize(ttreeRand));text(5,5,
    "time.obs = Random",
    cex = 1.5, pos = 4)
plot(ladderize(ttreeLAD));text(5,5,
    "time.obs = LAD",
    cex = 1.5, pos = 4)
layout(1)
par(parOrig)

# to get a fair sample of trees
    # let's increase ntrees

ttrees <- cal3TimePaleoPhy(
    cladogram,
    rangesCont,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate,
    ntrees = 9,
    plot = FALSE)

# let's compare nine of them at once in a plot

layout(matrix(1:9,3,3))
parOrig <- par(no.readonly = TRUE)
par(mar = c(0,0,0,0))
for(i in 1:9){
    plot(ladderize(ttrees[[i]]),
         show.tip.label = FALSE)
    }
layout(1)
par(parOrig)
# they are all a bit different!

# can plot the median diversity curve with multiDiv
multiDiv(ttrees)

# using node.mins
# let's say we have (molecular??) evidence that
    # node (5) is at least 1200 time-units ago
# to use node.mins, first need to drop any unshared taxa
droppers <- cladogram$tip.label[is.na(
    match(cladogram$tip.label,
           names(which(!is.na(rangesCont[,1])))
           )
    )
    ]
```

```
# and then drop those taxa
cladoDrop <- drop.tip(cladogram, droppers)

# now make vector same length as number of nodes
nodeDates <- rep(NA, Nnode(cladoDrop))
nodeDates[5] <- 1200
ttree <- cal3TimePaleoPhy(cladoDrop,
    rangesCont,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate,
    ntrees = 1,
    node.mins = nodeDates,
    plot = TRUE)

# example with time in discrete intervals
set.seed(444)
record <- simFossilRecord(p = 0.1,
     q = 0.1,
     nruns = 1,
     nTotalTaxa = c(30,40),
     nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record
    # with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.5)
# let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot = TRUE)
# Now let's use binTimeData to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length = 1)

# we can do something very similar for
    # the discrete time data (can be a bit slow)
likFun <- make_durationFreqDisc(rangesDisc)
spRes <- optim(
    parInit(likFun),
    likFun,
    lower = parLower(likFun),
    upper = parUpper(likFun),
    method = "L-BFGS-B",
    control = list(maxit = 1000000))
sProb <- spRes[[1]][2]

# but that's the sampling PROBABILITY per bin
    # NOT the instantaneous rate of change

# we can use sProb2sRate() to get the rate
    # We'll need to also tell it the int.length
sRate1 <- sProb2sRate(sProb,int.length = 1)

# we also need extinction rate and branching rate (see above)
    # need to divide by int.length...
divRate <- spRes[[1]][1]/1
```

```
# estimates that r = 0.3...
    # that's kind of low (simulated sampling rate is 0.5)
# Note: for real data, you may need to use an average int.length
    # (i.e. if intervals aren't all the same duration)
ttree <- bin_cal3TimePaleoPhy(cladogram,
    rangesDisc,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate1,
    ntrees = 1,
    plot = TRUE)
phyloDiv(ttree)

# can also force the appearance timings
    # not to be chosen stochastically
ttree1 <- bin_cal3TimePaleoPhy(cladogram,
    rangesDisc,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate1,
    ntrees = 1,
    nonstoch.bin = TRUE,
    plot = TRUE)
phyloDiv(ttree1)

# testing node.mins in bin_cal3TimePaleoPhy
ttree <- bin_cal3TimePaleoPhy(cladoDrop,
    rangesDisc,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate1,
    ntrees = 1,
    node.mins = nodeDates,
    plot = TRUE)
# with randres = TRUE
ttree <- bin_cal3TimePaleoPhy(cladoDrop,
    rangesDisc,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate1,
    ntrees = 1,
    randres = TRUE,
    node.mins = nodeDates,
    plot = TRUE)


# example with multiple values of anc.wt
ancWt <- sample(0:1,
    nrow(rangesDisc[[2]]),
    replace = TRUE)
names(ancWt) <- rownames(rangesDisc[[2]])
```

```
ttree1 <- bin_cal3TimePaleoPhy(cladogram,
    rangesDisc,
    brRate = divRate,
    extRate = divRate,
    sampRate = sRate1,
    ntrees = 1,
    anc.wt = ancWt,
    plot = TRUE)
```

`cladogeneticTraitCont` *Simulate Cladogenetic Trait Evolution*

Description

This function simulates trait evolution at each speciation/branching event in a matrix output from `simFossilRecord`, after transformation with `fossilRecord2fossilTaxa`.

Usage

```
cladogeneticTraitCont(taxa, rate = 1, meanChange = 0, rootTrait = 0)
```

Arguments

`taxa`	A five-column matrix of taxonomic data, as output by `fossilRecord2fossilTaxa` after simulation with `simFossilRecord` (or via the deprecated function `simFossilTaxa`)
`rate`	rate of trait change; variance of evolutionary change distribution per speciation event
`meanChange`	Mean change per speciation event. Default is 0; change to simulate 'active' speciational trends, where the expected change at each speciational event is non-zero.
`rootTrait`	The trait value of the first taxon in the dataset; set to 0 by default.

Details

This function simulates continuous trait evolution where change occurs under a Brownian model, but only at events that create new distinct morphotaxa (i.e. species as recognized in the fossil record), either branching events or anagenesis (pseudospeciation). These are the types of morphological differentiation which can be simulated in the function `simFossilRecord`. This is sometimes referred to as cladogenetic or speciation trait evolution and is related to Punctuated Equilibrium theory. Anagenetic shifts are not cladogenetic events per se (no branching!), so perhaps the best way to this of this function is it allows traits to change anytime `simFossilRecord` created a new 'morphotaxon' in a simulation.

Importantly, trait changes only occur at the base of 'new' species, thus allowing cladogenetic trait evolution to be asymmetrical at branching points: i.e. only one branch actually changes position

in trait-space, as expected under a budding cladogenesis model. This distinction is important as converting the taxa matrix to a phylogeny and simulating the trait changes under a 'speciational' tree-transformation would assume that divergence occurred on both daughter lineages at each node. (This has been the standard approach for simulating cladogenetic trait change on trees).

Cryptic taxa generated with `prop.cryptic` in `simFossilRecord` will not differ at all in trait values. These species will all be identical.

See this link for additional details: http://nemagraptus.blogspot.com/2012/03/simulating-budding-cladogenetict
html

Value

Returns a vector of trait values for each taxon, with value names being the taxa IDs (column 1 of the input) with a 't' pasted (as with `rtree` in the ape library).

Author(s)

David W. Bapst

See Also

simFossilRecord,

This function is similar to Brownian motion simulation functions such as `rTraitCont` in ape, `sim.char` in geiger and `fastBM` in phytools.

See also unitLengthTree in this package and `speciationalTree` in the package geiger. These are tree transformation functions; together with BM simulation functions, they would be expected to have a similar effect as this function (when cladogenesis is 'bifurcating' and not 'budding'; see above).

Examples

```
set.seed(444)
record <- simFossilRecord(
   p = 0.1, q = 0.1,
   nruns = 1,
   nTotalTaxa = c(30, 1000),
   plot = TRUE)
taxa <- fossilRecord2fossilTaxa(record)
trait <- cladogeneticTraitCont(taxa)
tree <- taxa2phylo(taxa)
plotTraitgram(trait, tree,
   conf.int = FALSE)

#with cryptic speciation
record <- simFossilRecord(
   p = 0.1, q = 0.1,
   prop.cryptic = 0.5,
   nruns = 1,
```

```
   nTotalTaxa = c(30, 1000),
   plot = TRUE)
taxa <- fossilRecord2fossilTaxa(record)
trait <- cladogeneticTraitCont(taxa)
tree <- taxa2phylo(taxa)
plotTraitgram(trait, tree,
   conf.int = FALSE)
```

---

communityEcology *Miscellaneous Functions for Community Ecology*

Description

This is just a small collection of miscellaneous functions that may be useful, primarily for community ecology analyses, particularly for paleoecological data. They are here mainly for pedagogical reasons (i.e. for students) as they don't appear to be available in other ecology-focused packages.

Usage

```
pairwiseSpearmanRho(
  x,
  dropAbsent = "bothAbsent",
  asDistance = FALSE,
  diag = NULL,
  upper = NULL,
  na.rm = FALSE
)

HurlbertPIE(x, nAnalyze = Inf)
```

Arguments

x	The community abundance matrix. Taxonomic units are assumed to be the columns and sites (samples) are assumed to be the rows, for both functions. The abundances can be absolute counts of specimens for particular taxa in each sample, or it can be proportional (relative) abundances, where all taxon abundances at a site are divided by the total number of specimens collected at that site. For function `pairwiseSpearmanRho`, the input `x` should be a matrix with two dimensions. For `HurlbertPIE`, a vector (an object with only one dimension) will be treated as if it was matrix with a single row and number of columns equal to its length.
dropAbsent	Should absent taxa be dropped? Must be one of either: `dropAbsent = 'bothAbsent'` (drop taxa absent in both sites for a given pairwise comparison), `dropAbsent = 'eitherAbsent'` (drop taxa absent in either site), or `dropAbsent = 'noDrop'` (drop none of the taxa absent in either site). The default `dropAbsent = 'bothAbsent'` is recommended; see examples.

asDistance	Should the rho coefficients be rescaled on a scale similar to dissimilarity metrics, i.e. bounded 0 to 1, with 1 representing maximum dissimilarity (i.e. a Spearman rho correlation of -1)? (Note that dissimilarity = (1 - rho) / 2)
diag	Should the diagonal of the output distance matrix be included?
upper	Should the upper triangle of the output distance matrix be included?
na.rm	Should taxa listed with NA values be dropped from a pair-wise site comparison? If na.rm = FALSE, the returned value for that site pair will be NA if NAs are present.
nAnalyze	Allows users to select that PIE be calculated only on the nAnalyze most-abundant taxa in a site sample. nAnalyze must be a vector of length = 1, consisting of a whole-number value greater than 1. By default, nAnalyze = Inf so all taxa are accepted. Note that if there are less taxa in a sample than nAnalyze, the number present will be used.

Details

pairwiseSpearmanRho returns Spearman rho correlation coefficients based on the rank abundances of taxa (columns) within sites (rows) from the input matrix, by internally wrapping the function cor.test. It allows for various options that automatically allow for dropping taxa not shared between two sites (the default), as well as several other options. This allows the rho coefficient to behave like the Bray-Curtis distance, in that it is not affected by the number of taxa absent in both sites.

pairwiseSpearmanRho can also rescale the rho coefficients with (1-rho)/2 to provide a measure similar to a dissimilarity metric, bounded between 0 and 1. This function was written so several arguments would be in a similar format to the vegan library function vegdist. If used to obtain rho rescaled as a dissimilarity, the default output will be the lower triangle of a distance matrix object, just as is returned by default by vegdist. This behavior can be modified via the arguments for including the diagonal and upper triangle of the matrix. Otherwise, a full matrix is returned (by default) if the asDistance argument is not enabled.

HurlbertPIE provides the 'Probability of Interspecific Encounter' metric for relative community abundance data, a commonly used metric for evenness of community abundance data based on derivations in Hurlbert (1971). An optional argument allows users to apply Hurlbert's PIE to only a subselection of the most abundant taxa.

Value

pairwiseSpearmanRho will return either a full matrix (the default) or (if asDistance is true, a distance matrix, with only the lower triangle shown (by default). See details.

HurlbertPIE returns a named vector of PIE values for the input data.

Author(s)

David W. Bapst

References

Hurlbert, S. H. 1971. The *nonconcept* of species diversity: a critique and alternative parameters. *Ecology* 52(4):577-586.

See Also

twoWayEcologyCluster; example dataset: kanto

Examples

```
# let's load some example data:
# a classic dataset collected by Satoshi & Okido from the Kanto region

data(kanto)

rhoBothAbsent <- pairwiseSpearmanRho(kanto,dropAbsent = "bothAbsent")

#other dropping options
rhoEitherAbsent <- pairwiseSpearmanRho(kanto,dropAbsent = "eitherAbsent")
rhoNoDrop <- pairwiseSpearmanRho(kanto,dropAbsent = "noDrop")

#compare
layout(1:3)
lim <- c(-1,1)
plot(rhoBothAbsent, rhoEitherAbsent, xlim = lim, ylim = lim)
abline(0,1)
plot(rhoBothAbsent, rhoNoDrop, xlim = lim, ylim = lim)
abline(0,1)
plot(rhoEitherAbsent, rhoNoDrop, xlim = lim, ylim = lim)
abline(0,1)
layout(1)

#using dropAbsent = "eitherAbsent" reduces the number of taxa so much that
# the number of taxa present drops too low to be useful
#dropping none of the taxa restricts the rho measures to high coefficients
# due to the many shared 0s for absent taxa

#############

# Try the rho coefficients as a rescaled dissimilarity
rhoDist <- pairwiseSpearmanRho(kanto,asDistance = TRUE,dropAbsent = "bothAbsent")

# What happens if we use these in typical distance matrix based analyses?

# Cluster analysis
clustRes <- hclust(rhoDist)
plot(clustRes)

# Principle Coordinates Analysis
pcoRes <- pcoa(rhoDist,correction = "lingoes")
scores <- pcoRes$vectors
#plot the PCO
plot(scores,type = "n")
text(labels = rownames(kanto),scores[,1],scores[,2],cex = 0.5)

##################################
```

```
# measuring evenness with Hurlbert's PIE

kantoPIE <- HurlbertPIE(kanto)

#histogram
hist(kantoPIE)
#evenness of the kanto data is fairly high

#barplot
parX <- par(mar = c(7,5,3,3))
barplot(kantoPIE,las = 3,cex.names = 0.7,
ylab = "Hurlbert's PIE",ylim = c(0.5,1),xpd = FALSE)
par(parX)

#and we can see that the Tower has extremely low unevenness
#...overly high abundance of ghosts?

# NOTE it doesn't matter whether we use absolute abundances
# or proportional (relative) abundances
kantoProp<-t(apply(kanto,1,function(x) x/sum(x)))
kantoPropPIE <- HurlbertPIE(kantoProp)
identical(kantoPIE,kantoPropPIE)

#let's look at evenness of 5 most abundant taxa
kantoPIE_5 <- HurlbertPIE(kanto,nAnalyze = 5)

#barplot
parX <- par(mar = c(7,5,3,3))
barplot(kantoPIE_5,las = 3,cex.names = 0.7,
ylab = "Hurlbert's PIE for 5 most abundant taxa",ylim = c(0.5,1),xpd = FALSE)
par(parX)
```

`compareTimescaling` *Comparing the Time-Scaling of Trees*

Description

These functions take two trees and calculate the changes in node ages (for `compareNodeAges`) for shared clades or terminal branch lengths leading to shared tip taxa (for `compareTermBranches`).

Usage

```
compareNodeAges(tree1, tree2, dropUnshared = FALSE)

compareTermBranches(tree1, tree2)
```

Arguments

tree1	A time-scaled phylogeny of class phylo
tree2	A time-scaled phylogeny of class phylo; for compareNodeAges, tree2 can also be an object of class multiPhylo composed of multiple phylogenies. See below.
dropUnshared	If TRUE, nodes not shared across all input trees are dropped from the final output for compareNodeAge. This argument has no effect if tree2 is a single phylogeny (a phylo-class object).

Details

For their most basic usage, these functions compare the time-scaling of two trees. Any taxa not-shared on both trees are dropped before analysis, based on tip labels.

As with many paleotree functions, calculations relating to time on trees are done with respect to any included $root.time elements. If these are not present, the latest tip is assumed to be at the present day (time = 0).

The function compareNodeAges calculates the changes in the clade ages among those clades shared by the two trees, relative to the first tree in absolute time. For example, a shift of +5 means the clade originates five time-units later in absolute time on the second tree, while a shift of -5 means the clade originated five time-units *prior* on the second tree.

For compareNodeAges, if tree2 is actually a multiPhylo object composed of multiple phylogenies, the output will be a matrix, with each row representing a different tree and each column a different clade shared between at least some subset of the trees in tree2 and the tree in tree1. values in the matrix are the changes in clade ages between from tree1 (as baseline) to tree2, with NA values representing a clade that is not contained in the tree represented by that row (but is contained in tree1 and at least one other tree in tree2). The matrix can be reduced to only those clades shared by all trees input via the argument dropUnshared. Note that this function distinguishes clades based on their shared taxa, and cannot so infer that two clades might be identical if it were not for single taxon within the crown of one considered clade, despite that such a difference should probably have no effect on compare a node divergence date. Users should consider their dataset for such scenarios prior to application of compareNodeAges, perhaps by dropping all taxa not included in all other trees to be considered (this is NOT done by this function).

compareTermBranches calculates the changes in the terminal branch lengths attached to tip taxa shared by the two trees, relative to the first tree. Thus, a shift of +5 means that this particular terminal taxon is connected to a terminal branch which is five time-units longer.

Value

compareTermBranches returns a vector of temporal shifts for terminal branches with the shared tip names as labels.

For the function compareNodeAges, if both tree1 and tree2 are single trees, outputs a vector of temporal shifts for nodes on tree2 with respect to tree1. If tree2 is multiple trees, then a matrix is output, with each row representing each tree in tree2 (and carrying the name of each tree, if any is given). The values are temporal shifts for each tree in tree2 with respect to tree1. For either case, the column names or element names (for a vector) are the sorted taxon names of the particular clade, the dates of which are given in that column. See above for more details. These names can be very long when large trees are considered.

See Also

dateNodes, taxa2phylo, phyloDiv

Examples

```
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
#get the true tree
tree1 <- taxa2phylo(taxa)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r = 0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot = TRUE)
#Now let's try timePaleoPhy using the continuous range data
tree2 <- timePaleoPhy(cladogram,rangesCont,type = "basic")
#let's look at the distribution of node shifts
hist(compareNodeAges(tree1,tree2))
#let's look at the distribution of terminal branch lengths
hist(compareTermBranches(tree1,tree2))

#testing ability to compare multiple trees with compareNodeAges
trees <- cal3TimePaleoPhy(cladogram,rangesCont,brRate = 0.1,extRate = 0.1,
    sampRate = 0.1,ntrees = 10)
nodeComparison <- compareNodeAges(tree1,trees)
#plot it as boxplots for each node
boxplot(nodeComparison,names = NULL);abline(h = 0)
#plot mean shift in node dates
abline(h = mean(apply(nodeComparison,2,mean,na.rm = TRUE)),lty = 2)

#just shifting a tree back in time
set.seed(444)
tree1 <- rtree(10)
tree2 <- tree1
tree1$root.time <- 10
compareNodeAges(tree1,tree2)
compareTermBranches(tree1,tree2)
```

---

constrainParPaleo *Constrain Parameters for a Model Function from paleotree*

Description

This function constrains a model to make submodels with fewer parameters, using a structure and syntax taken from the function constrain in Rich Fitzjohn's package diversitree.

Usage

```
constrainParPaleo(f, ..., formulae = NULL, names = parnames(f), extra = NULL)
```

Arguments

f	A function to constrain. This function must be of the S3 class paleotreeFunc and have all necessary attributes expected of that class, which include parameter names and upper and lower bounds. As I have deliberately not exported the function which creates this class, it should be impossible for non-advanced users to obtain such objects easily without using one of the make functions, which automatically output a function of the appropriate class and attributes.
...	Formulae indicating how the function should be constrained. See details and examples for lengthy discussion.
formulae	Optional list of constraints, possibly in addition to those in ...
names	Optional Character vector of names, the same length as the number of parameters in x. Use this only if parnames does not return a vector for your function. Generally this should not be used. DWB: This argument is kept for purposes of keeping the function as close to the parent as possible but, in general, should not be used because the input function must have all attributes expected of class paleotreeFunc, including parameter names.
extra	Optional vector of additional names that might appear on the RHS of constraints but do not represent names in the function's argnames. This can be used to set up dummy variables (example coming later).

Details

This function is based on (but does not depend on) the function constrain from the package diversitree. Users should refer to this parent function for more detailed discussion of model constraint usage and detailed examples.

The parent function was forked to add functionality necessary for dealing with the high parameter count models typical to some paleontological analyses, particularly the inverse survivorship method. This necessitated that the new function be entirely separate from its parent. Names of functions involved (both exported and not) have been altered to avoid overlap in the package namespaces. Going forward, the paleotree package maintainer (Bapst) will try to be vigilant with respect to changes in constrain in the original package, diversitree.

Useful information from the diversitree manual (11/01/13):

"If f is a function that takes a vector x as its first argument, this function returns a new function that takes a shorter vector x with some elements constrained in some way; parameters can be fixed to particular values, constrained to be the same as other parameters, or arbitrary expressions of free parameters."

In general, formulae should be of the structure:

LHS ~ RHS

...where the LHS is the 'Parameter We Want to Constrain' and the RHS is whatever we are constraining the LHS to, usually another parameter. LHS and RHS are the 'left-hand side' and 'right-hand side' respectively (which I personally find obscure).

Like the original `constrain` function this function is based on, this function cannot remove constraints previously placed on a model object and there may be cases in which the constrained function may not make sense, leading to an error. The original function will sometimes issue nonsensical functions with an incorrect number/names of parameters if the parameters to be constrained are given in the wrong order in formulae.

Differences from the original `constrain` function from `diversitree`:

This forked `paleotree` version of constrain has two additional features, both introduced to aid in constraining models with a high number of repetitive parameters. (I did not invent these models, so don't shoot the messenger.)

First, it allows nuanced control over the constraining of many parameters simultaneously, using the `all` and `match` descriptors. This system depends on parameters being named as such: `name.group1.group2.group3` and so on. Each 'group' is reference to a system of groups, perhaps referring to a time interval, region, morphology, taxonomic group or some other discrete characterization among the data (almost all functions envisioned for paleotree are for per-taxon diversification models). The number of group systems is arbitrary, and may be from zero to a very large number; it depends on the 'make' function used and the arguments selected by the user. For example, the parameter `x.1` would be for the parameter `x` in the first group of the first group system (generally a time interval for most `paleotree` functions). For a more complicated exampled, with the parameter `x.1.3.1`, the third group for the second group system (perhaps this taxonomic data point has a morphological feature not seen in some other taxa) and group 1 of the third group system (maybe biogeographic region 1? The possibilities are endless depending on user choices!).

The `all` option work like so: if `x.all ~ x.1` is given as a formulae, then all `x` parameters will be constrained to equal `x.1`. For example, if there is `x.1`, `x.2`, `x.3` and `x.4` parameters for a model, `x.all ~ x.1` would be equivalent to individually giving the formulae `x.2~x.1`, `x.3~x.1` and `x.4~x.1`. This means that if there are many parameters of a particular type (say, 50 `x` parameters) it is easy to constrain all with a short expression. It is not necessary that the The `all` term can be used anywhere in the name of the parameter in a formulae, including to make all parameters for a given `group` the same. Furthermore, the LHS and RHS don't need to be same parameter group, and both can contain `all` statements, even *multiple* `all` statements. Consider these examples, each of which are legal, acceptable uses:

`x.all ~ y.1` Constrains all values of the parameter `x` for every group to be equal to the single value for the parameter `y` for group 1 (note that there's only a single set of groups).

`all.1 ~ x.1` Constrains all parameters for the first group to equal each other, here arbitrary named `x.1`. For example, if there is parameters named `x.1`, `y.1` and `z.1`, all will be constrained to be equal to a single parameter value.

`x.all.all ~ y.2.3` Constrains all values for `x` in any and all groups to equal the single value for `y` for group 2 (of system 1) and group 3 (of system 2).

`x.all ~ y.all` Constrains all values of `x` for every group and `y` for every group to be equal to a *single* value, which by default will be reported as `y.1`

The `match` term is similar, allowing parameter values from the same group to be quickly matched and made equivalent. These `match` terms must have a matching (cue laughter) term both in the corresponding LHS and RHS of the formula. For example, consider `x.match ~ y.match` where there are six parameters: `x.1`, `x.2`, `x.3`, `y.1`, `y.2` and `y.3`. This will effectively constrain `x.1 ~ y.1`, `x.2 ~ y.2` and `x.3 ~ y.3`. This is efficient for cases where we have some parameters that we often treat as equal. For example, in paleontology, we sometimes make a simplifying assumption that birth and death rates are equal in multiple time intervals. Some additional legal examples are:

`x.match.1 ~ y.match.1` This will constrain only parameters of `x` and `y` to to equal each other if they both belong to the same group for the first group system AND belong to group 1 of the first group.

`all.match. ~ x.match` This will constrain all named parameters in each group to equal each other; for example, if there are parameters `x.1`, `y.1`, `z.1`, `x.2`, `y.2` and `z.2`, this will constrain them such that `y.1 ~ x.1`, `z.1 ~ x.1`, `y.2 ~ x.2` and `z.2 ~ x.2`, leaving `x.1` and `x.2` as the only parameters effectively.

There are two less fortunate qualities to the introduction of the above terminology.

Unfortunately, this package author apologizes that his programming skills are not good enough to allow more complex sets of constraints, as would be typical with the parent function, when `all` or `match` terms are included. For example, it would not be legal to attempt to constraint `y.all ~ x.1 / 2`, where the user presumably is attempting to constrain all y values to equal the x parameter to equal half of the x parameter for group 1. This will not be parsed as such and should return an error. However, there are workarounds, but they require using `constrainParPaleo` more than once. For the above example, a user could first use `y.all ~ y.1` constraining all y values to be equal. Then a user could constrain with the formula `y.1 ~ x.1 / 2` which would then constrain `y.1` (and all the y values constrained to equal it) to be equal to the desired fraction.

Furthermore, this function expects that parameter names don't already have period-separated terms that are identical to `all` or `match`. No function in `paleotree` should produce such natively. If such were to occur, perhaps by specially replacing parameter names, `constrainParPaleo` would confuse these terms for the specialty terms described here.

Secondly, this altered version of constrain handles the parameter bounds included as attributes in functions output by the various 'make' functions. This means that if `x.1 ~ y.1` is given, `constrainParPaleo` will test if the bounds on `x.1` and `y.1` are the same. If the bounds are not the same, `constrainParPaleo` will return an error. This is important, as some models in paleotree may make a parameter a rate (bounded zero to some value greater than one) or a probability (bounded zero to one), depending on user arguments. Users may not realize these differences and, in many cases, constraining a rate to equal a probability is nonsense (absolute poppycock!). If a user really wishes to constrain two parameters with different bounds to be equal (I have no idea why anyone would want to do this), they can use the parameter bound replacement functions described in `modelMethods` to set the parameter bounds as equal. Finally, once parameters with the same bounds are constrained, the output has updated bounds that reflect the new set of parameters for the new constrained function.

Value

Modified from the `diversitree` manual: This function returns a constrained function that can be passed through to the optimization functions of a user's choice, such as `optim`, `find.mle` in `diversitree` or `mcmc`. It will behave like any other function. However, it has a modified `class` attribute so that some methods will dispatch differently: `parnames`, for example, will return the names of the parameters of the constrained function and `parInit` will return the initial values for those same constrained set of parameters. All arguments in addition to `x` will be passed through to the original function `f`.

Additional useful information from the `diversitree` manual (11/01/13):

For help in designing constrained models, the returned function has an additional argument `pars.only`, when this is `TRUE` the function will return a named vector of arguments rather than evaluate the function (see Examples).

Author(s)

This function (and even this help file!) was originally written by Rich Fitzjohn for his library `diversitree`, and subsequently rewritten and modified by David Bapst.

References

FitzJohn, R. G. 2012. `diversitree`: comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution* 3(6):1084-1092.

See Also

As noted above, this function is strongly based on (but does not depend on) the function `constrain` from the library `diversitree`.

Examples

```
# simulation example with make_durationFreqCont, with three random groups
set.seed(444)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = 0
    )
taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r = 0.5)
# create a groupings matrix
grp1 <- matrix(
    sample(1:3,nrow(taxa),replace = TRUE), , 1)
likFun <- make_durationFreqCont(rangesCont, groups = grp1)

# can constrain both extinction rates to be equal
constrainFun <- constrainParPaleo(likFun, q.2 ~ q.1)

#see the change in parameter names and bounds
parnames(likFun)
parnames(constrainFun)
parbounds(likFun)
parbounds(constrainFun)

# some more ways to constrain stuff!

#constrain all extinction rates to be equal
constrainFun <- constrainParPaleo(likFun, q.all ~ q.1)
parnames(constrainFun)

#constrain all rates for everything to be a single parameter
constrainFun <- constrainParPaleo(likFun, r.all ~ q.all)
parnames(constrainFun)

#constrain all extinction rates to be equal & all sampling to be equal
```

```
constrainFun <- constrainParPaleo(likFun, q.all ~ q.1, r.all ~ r.1)
parnames(constrainFun)

#similarly, can use match.all to make all matching parameters equal each other
constrainFun <- constrainParPaleo(likFun, match.all ~ match.all)
parnames(constrainFun)

#Constrain rates in same group to be equal
constrainFun <- constrainParPaleo(likFun, r.match ~ q.match)
parnames(constrainFun)
```

`createMrBayesConstraints` *Transform a Topology into a Set of Constraint Commands for MrBayes*

Description

Takes a phylogeny in the form of an object of class `phylo` and outputs a set of topological constraints for *MrBayes* as a set of character strings, either printed in the R console or in a named text file, which can be used as commands in the *MrBayes* block of a NEXUS file for use with (you guessed it!) *MrBayes*.

Usage

```
createMrBayesConstraints(
  tree,
  partial = TRUE,
  file = NULL,
  includeIngroupConstraint = FALSE
)
```

Arguments

`tree`	An object of class `phylo`.
`partial`	If `TRUE` (the default), then constraints will be defined as partial constraints with respect to the rest of the taxa in the input `tree`. If `FALSE`, constraints will be defined as hard clade membership constraints (i.e. no additional taxa will be allowed to belong to the nodes present in the observed tree. Depending on your analysis, `partial = TRUE` may require additionally defining an outgroup.
`file`	Filename (possibly with path) as a character string to a file which will be overwritten with the output constraint lines. If `NULL`, constraint lines are printed in the console.
`includeIngroupConstraint`	When writing the `prset` line, should a group named 'ingroup' be included, which presumes an ingroup constraint was defined by the user for sake or rooting the tree? This is mainly used for use with `paleotree` function `createMrBayesTIpDatingNexus` for automating the construction of tip-dating analyses, which must constrain the ingroup.

Details

partial = TRUE may be useful if the reason for using createMrBayesConstraints is to constrain a topology containing some of the taxa in an analysis, while allowing other taxa to freely vary. For example, Slater (2013) constrained an analysis so extant taxon relationships were held constant, using a molecular-based topology, while allowing fossil taxa to freely vary relative to their morphological character data.

Value

If argument file is NULL, then the constrain commands are output as a series of character strings.

Author(s)

David W. Bapst, with some inspiration from Graham Slater. This code was produced as part of a project funded by National Science Foundation grant EAR-1147537 to S. J. Carlson.

References

Slater, G. J. 2013. Phylogenetic evidence for a shift in the mode of mammalian body size evolution at the Cretaceous-Palaeogene boundary. *Methods in Ecology and Evolution* 4(8):734-744.

See Also

createMrBayesTipDatingNexus, createMrBayesTipCalibrations

Examples

```
set.seed(444)
tree <- rtree(10)
createMrBayesConstraints(tree)
createMrBayesConstraints(tree,partial = FALSE)

## Not run:

createMrBayesConstraints(tree,file = "topoConstraints.txt")


## End(Not run)
```

---

createMrBayesTipCalibrations

Construct A Block of Tip Age Calibrations for Use with Tip-Dating Analyses in MrBayes

---

Description

Takes a set of tip ages (in several possible forms, see below), and outputs a set of tip age calibrations for use with tip-dating analyses (sensu Zhang et al., 2016) in the popular phylogenetics program *MrBayes*. These calibrations are printed as a set of character strings, as well as a line placing an offset exponential prior on the tree age, either printed in the R console or in a named text file, which can be used as commands in the *MrBayes* block of a NEXUS file for use with (you guessed it!) *MrBayes*.

Usage

```
createMrBayesTipCalibrations(
  tipTimes,
  ageCalibrationType,
  whichAppearance = "first",
  treeAgeOffset,
  minTreeAge = NULL,
  collapseUniform = TRUE,
  anchorTaxon = TRUE,
  file = NULL
)
```

Arguments

`tipTimes` This input may be either: (a) a `timeList` object, consisting of a `list` of `length = 2`, composed of a table of interval upper and lower time boundaries (i.e., the earlier and latter bounds of the intervals) and a table of first and last intervals for taxa, or (b) a matrix with row names corresponding to taxon names, matching those names listed in the MrBayes block, with either one, two or four columns containing ages (respectively) for point occurrences with precise dates (for a single column), uncertainty bounds on a point occurrence (for two columns), or uncertainty bounds on the first and last occurrence (for four columns). Note that precise first and last occurrence dates should not be entered as a two column matrix, as this will instead be interpreted as uncertainty bounds on a single occurrence. Instead, either select which you want to use for tip-dates and give a one-column matrix, or repeat (and collate) the columns, so that the first and last appearances has uncertainty bounds of zero.

`ageCalibrationType`
This argument decides how age calibrations are defined, and currently allows for four options: `"fixedDateEarlier"` which fixes tip ages at the earlier (lower) bound for the selected age of appearance (see argument `whichAppearance` for how that selection is made), `"fixedDateLatter"` which fixes the date to the latter (upper) bound of the selected age of appearance, `"fixedDateRandom"` which fixes tips to a date that is randomly drawn from a uniform distribution bounded by the upper and lower bounds on the selected age of appearance, or (the recommended option) `"uniformRange"` which places a uniform prior on the age of the tip, bounded by the latest and earliest (upper and lower) bounds on the the selected age.

`whichAppearance`
: Which appearance date of the taxa should be used: their `'first'` or their `'last'` appearance date? The default option is to use the 'first' appearance date. Note that use of the last appearance date means that tips will be constrained to occur before their last occurrence, and thus could occur long after their first occurrence (!).

`treeAgeOffset`
: A parameter given by the user controlling the offset between the minimum and expected tree age prior. mean tree age for the offset exponential prior on tree age will be set to the minimum tree age, plus this offset value. Thus, an offset of 10 million years would equate to a prior assuming that the expected tree age is around 10 million years before the minimum age.

`minTreeAge`
: if `NULL` (the default), then `minTreeAge` will be set as the oldest date among the tip age used (those used being determine by user choices (or oldest bound on a tip age). Otherwise, the user can supply their own minimum tree, which must be greater than whatever the oldest tip age used is.

`collapseUniform`
: MrBayes won't accept uniform age priors where the maximum and minimum age are identical (i.e. its actually a fixed age). Thus, if this argument is `TRUE` (the default), this function will treat any taxon ages where the maximum and minimum are identical as a fixed age, and will override setting `ageCalibrationType = "uniformRange"` for those dates. All taxa with their ages set to fixed by the behavior of `anchorTaxon` or `collapseUniform` are returned as a list within a commented line of the returned MrBayes block.

`anchorTaxon`
: This argument may be a logical (default is `TRUE`, or a character string of length = 1. This argument has no effect if `ageCalibrationType` is not set to `"uniformRange"`, but the argument may still be evaluated. If `ageCalibrationType = "uniformRange"`, MrBayes will do a tip-dating analysis with uniform age uncertainties on all taxa (if such uncertainties exist; see `collapseUniform`). However, MrBayes does not record how each tree sits on an absolute time-scale, so if the placement of *every* tip is uncertain, lining up multiple dated trees sampled from the posterior (where each tip's true age might differ) could be a nightmare to back-calculate, if not impossible. Thus, if `ageCalibrationType = "uniformRange"`, and there are no tip taxa given fixed dates due to `collapseUniform` (i.e. all of the tip ages have a range of uncertainty on them), then a particular taxon will be selected and given a fixed date equal to its earliest appearance time for its respective `whichAppearance`. This taxon can either be indicated by the user or instead the first taxon listed in `tipTimes` will be arbitrary selected. All taxa with their ages set to fixed by the behavior of `anchorTaxon` or `collapseUniform` are returned as a list within a commented line of the returned MrBayes block.

`file`
: Filename (possibly with path) as a character string to a file which will be overwritten with the output tip age calibrations. If `NULL`, tip calibration commands are output to the console.

Details

Beware: some combinations of arguments might not make sense for your data.

(But that's always true, is it not?)

Value

If argument `file` is `NULL`, then the tip age commands are output as a series of character strings.

All taxa with their ages set to fixed by the behavior of `anchorTaxon` or `collapseUniform` are returned as a list within a commented line of the returned MrBayes block.

Author(s)

David W. Bapst. This code was produced as part of a project funded by National Science Foundation grant EAR-1147537 to S. J. Carlson.

References

Zhang, C., T. Stadler, S. Klopfstein, T. A. Heath, and F. Ronquist. 2016. Total-Evidence Dating under the Fossilized Birth-Death Process. *Systematic Biology* 65(2):228-249.

See Also

`createMrBayesConstraints`, `createMrBayesTipDatingNexus`

Examples

```
# load retiolitid dataset
data(retiolitinae)

# uniform prior, with a 10 million year offset for
# the expected tree age from the earliest first appearance

createMrBayesTipCalibrations(
   tipTimes = retioRanges,
   whichAppearance = "first",
  ageCalibrationType = "uniformRange",
  treeAgeOffset = 10)

# fixed prior, at the earliest bound for the first appearance

createMrBayesTipCalibrations(
   tipTimes = retioRanges,
   whichAppearance = "first",
  ageCalibrationType = "fixedDateEarlier",
  treeAgeOffset = 10
  )

# fixed prior, sampled from between the bounds on the last appearance
# you should probably never do this, fyi

createMrBayesTipCalibrations(
   tipTimes = retioRanges,
   whichAppearance = "first",
  ageCalibrationType = "fixedDateRandom",
  treeAgeOffset = 10
```

```
  )


## Not run:

createMrBayesTipCalibrations(
   tipTimes = retioRanges,
   whichAppearance = "first",
  ageCalibrationType = "uniformRange",
  treeAgeOffset = 10,
  file = "tipCalibrations.txt"
  )


## End(Not run)
```

`createMrBayesTipDatingNexus` *Construct a Fully Formatted NEXUS Script for Performing Tip-Dating Analyses With MrBayes*

Description

This function is meant to expedite the creation of NEXUS files formatted for performing tip-dating analyses in the popular phylogenetics software *MrBayes*, particularly clock-less tip-dating analyses executed with 'empty' morphological matrices (i.e. where all taxa are coded for a single missing character), although a pre-existing morphological matrix can also be input by the user (see argument `origNexusFile`). Under some options, this pre-existing matrix may be edited by this function. The resulting full NEXUS script is output as a set of character strings either printed to the R console, or output to file which is then overwritten.

Usage

```
createMrBayesTipDatingNexus(
  tipTimes,
  outgroupTaxa = NULL,
  treeConstraints = NULL,
  ageCalibrationType,
  whichAppearance = "first",
  treeAgeOffset,
  minTreeAge = NULL,
  collapseUniform = TRUE,
  anchorTaxon = TRUE,
  newFile = NULL,
  origNexusFile = NULL,
  parseOriginalNexus = TRUE,
  createEmptyMorphMat = TRUE,
```

```
  orderedChars = NULL,
  morphModel = "strong",
  morphFiltered = "parsInf",
  runName = NULL,
  ngen = "100000000",
  doNotRun = FALSE,
  autoCloseMrB = FALSE,
  cleanNames = TRUE,
  printExecute = TRUE
)
```

Arguments

tipTimes	This input may be either: (a) a timeList object, consisting of a list of length = 2, composed of a table of interval upper and lower time boundaries (i.e., the earlier and latter bounds of the intervals) and a table of first and last intervals for taxa, or (b) a matrix with row names corresponding to taxon names, matching those names listed in the MrBayes block, with either one, two or four columns containing ages (respectively) for point occurrences with precise dates (for a single column), uncertainty bounds on a point occurrence (for two columns), or uncertainty bounds on the first and last occurrence (for four columns). Note that precise first and last occurrence dates should not be entered as a two column matrix, as this will instead be interpreted as uncertainty bounds on a single occurrence. Instead, either select which you want to use for tip-dates and give a one-column matrix, or repeat (and collate) the columns, so that the first and last appearances has uncertainty bounds of zero.
outgroupTaxa	A vector of type 'character', containing taxon names designating the outgroup. All taxa not listed in the outgroup will be constrained to be a monophyletic ingroup, for sake of rooting the resulting dated tree. Either treeConstraints or outgroupTaxa must be defined, but *not both*. If the outgroup-ingroup split is not present on the supplied treeConstraints, add that split to treeConstraints manually.
treeConstraints	An object of class phylo, from which (if treeConstraints is supplied) the set topological constraints are derived, as as described for argument tree for function createMrBayesConstraints. Either treeConstraints or outgroupTaxa must be defined, but *not both*. If the outgroup-ingroup split is not present on the supplied treeConstraints, add that split to treeConstraints manually.
ageCalibrationType	This argument decides how age calibrations are defined, and currently allows for four options: "fixedDateEarlier" which fixes tip ages at the earlier (lower) bound for the selected age of appearance (see argument whichAppearance for how that selection is made), "fixedDateLatter" which fixes the date to the latter (upper) bound of the selected age of appearance, "fixedDateRandom" which fixes tips to a date that is randomly drawn from a uniform distribution bounded by the upper and lower bounds on the selected age of appearance, or (the recommended option) "uniformRange" which places a uniform prior on

the age of the tip, bounded by the latest and earliest (upper and lower) bounds on the the selected age.

`whichAppearance`

Which appearance date of the taxa should be used: their `'first'` or their `'last'` appearance date? The default option is to use the 'first' appearance date. Note that use of the last appearance date means that tips will be constrained to occur before their last occurrence, and thus could occur long after their first occurrence (!). In addition, `createMrBayesTipDatingNexus` allows for two options for this argument that are in addition to those offered by `createMrBayesTipCalibrations`. Both of these options will duplicate the taxa in the inputs multiple times, modifying their OTU labels, thus allowing multiple occurrences of long-lived morphotaxa to be listed as multiple OTUs arrayed across their stratigraphic duration. If `whichAppearance = "firstLast"`, taxa will be duplicated so each taxon is listed as occurring twice: once at their first appearance, and a second time at their last appearance. Note that if a taxon first and last appears in the same interval, and `ageCalibrationType = "uniformRange"`, then the resulting posterior trees may place the OTU assigned to the last occurrence before the first occurrence in temporal order (but the assignment, in that case, was entirely arbitrary). When `whichAppearance = "rangeThrough"`, each taxon will be duplicated into as many OTUs as each interval that a taxon ranges through (in a `timeList` format, see other `paleotree` functions), with the corresponding age uncertainties for those intervals. If the input `tipTimes` is not a list of `length = 2`, however, the function will return an error under this option.

`treeAgeOffset` A parameter given by the user controlling the offset between the minimum and expected tree age prior. mean tree age for the offset exponential prior on tree age will be set to the minimum tree age, plus this offset value. Thus, an offset of 10 million years would equate to a prior assuming that the expected tree age is around 10 million years before the minimum age.

`minTreeAge` if `NULL` (the default), then `minTreeAge` will be set as the oldest date among the tip age used (those used being determine by user choices (or oldest bound on a tip age). Otherwise, the user can supply their own minimum tree, which must be greater than whatever the oldest tip age used is.

`collapseUniform`

MrBayes won't accept uniform age priors where the maximum and minimum age are identical (i.e. its actually a fixed age). Thus, if this argument is `TRUE` (the default), this function will treat any taxon ages where the maximum and minimum are identical as a fixed age, and will override setting `ageCalibrationType = "uniformRange"` for those dates. All taxa with their ages set to fixed by the behavior of `anchorTaxon` or `collapseUniform` are returned as a list within a commented line of the returned MrBayes block.

`anchorTaxon` This argument may be a logical (default is `TRUE`, or a character string of length = 1. This argument has no effect if `ageCalibrationType` is not set to `"uniformRange"`, but the argument may still be evaluated. If `ageCalibrationType = "uniformRange"`, MrBayes will do a tip-dating analysis with uniform age uncertainties on all taxa (if such uncertainties exist; see `collapseUniform`). However, MrBayes does not record how each tree sits on an absolute time-scale, so if the placement of *every* tip is uncertain, lining up multiple dated trees sampled from the posterior

(where each tip's true age might differ) could be a nightmare to back-calculate, if not impossible. Thus, if `ageCalibrationType = "uniformRange"`, and there are no tip taxa given fixed dates due to `collapseUniform` (i.e. all of the tip ages have a range of uncertainty on them), then a particular taxon will be selected and given a fixed date equal to its earliest appearance time for its respective `whichAppearance`. This taxon can either be indicated by the user or instead the first taxon listed in `tipTimes` will be arbitrary selected. All taxa with their ages set to fixed by the behavior of `anchorTaxon` or `collapseUniform` are returned as a list within a commented line of the returned MrBayes block.

`newFile` Filename (possibly with path) as a character string leading to a file which will be overwritten with the output tip age calibrations. If `NULL`, tip calibration commands are output to the console.

`origNexusFile` Filename (possibly with path) as a character string leading to a NEXUS text file, presumably containing a matrix of character date formated for *MrBayes*. If supplied (it does not need to be supplied), the listed file is read as a text file, and concatenated with the *MrBayes* script produced by this function, so as to reproduce the original NEXUS matrix for executing in MrBayes. Note that the taxa in this NEXUS file are *NOT* checked against the user input `tipTimes` and `treeConstraints`, so it is up to the user to ensure the taxa are the same across the three data sources.

`parseOriginalNexus`

If `TRUE` (the default), the original NEXUS file is parsed and the taxon names listed within in the matrix are compared against the other inputs for matching (completely, across all inputs that include taxon names). Thus, it is up to the user to ensure the same taxa are found in all inputs. However, some NEXUS files may not parse correctly (particularly if character data for taxa stretches across more than a single line in the matrix). This may necessitate setting this argument to `FALSE`, which will instead do a straight scan of the NEXUS matrix without parsing it, and without checking the taxon names against other outputs. Some options for `whichAppearance` will not be available, however.

`createEmptyMorphMat`

If `origNexusFile` is not specified (implying there is no pre-existing morphological character matrix for this dataset), then an 'empty' NEXUS-formatted matrix will be appended to the set of *MrBayes* commands if this command is `TRUE` (the default). This 'empty' matrix will have each taxon in `tipTimes` coded for a single missing character (i.e., '?'). This allows tip-dating analyses with hard topological constraints, and ages determined entirely by the fossilized birth-death prior, with no impact from a presupposed morphological clock (thus a 'clock-less analysis').

`orderedChars` Should be a vector of numbers, indicating which characters should have their character-type in MrBayes changed to 'ordered'. If `NULL`, the default, then all characters will be treated as essentially unordered. No character ID should be listed that is higher than the number of characters in the matrix provided in `origNexusFile`. If `origNexusFile` is not provided, while `orderedChars` is defined, then an error will be returned.

`morphModel` This argument can be used to switch between two end-member models of morphological evolution in MrBayes, here named 'strong' and 'relaxed', for the

'strong assumptions' and 'relaxed assumptions' models described by Bapst et al. (2018, Syst. Biol.). The default is a model which makes very 'strong' assumptions about the process of morphological evolution, while the 'relaxed' alternative allows for considerably more heterogeneity in the rate of morphological evolution across characters, and in the forward and reverse transition rates between states. Also see argument `morphFiltered`.

`morphFiltered` This argument controls what type of filtering the input morphological data is assumed to have been collected under. The likelihood of the character data will be modified to take into account the apparent filtering (Lewis, 2001; Allman et al., 2010). The default value, `"parsInf"`, forces characters to be treated as if they were collected as part of a parsimony-based study, with constant characters and autapomorphies (characters that only differ in state in a single taxon unit) ignored or otherwise filtered out, and any such characters in the presented matrix will be ignored. `morphFiltered = "variable"` assumes that while constant characters are still filtered out (e.g. it is difficult or impossible to count the number of morphological characters that show no variation across a group), the autapomorphies were intentionally collected and included in the presented matrix. Thus, constant characters in the included matrix will be ignored, but autapomorphies will be considered.

`runName` The name of the run, used for naming the log files and MCMC output files. If not set, the name will be taken from the name given for outputting the NEXUS script (`newFile`). If `newFile` is not given, and `runName` is not set by the user, the default run name will be "new_run_paleotree".

`ngen` Number of generations to set the MCMCMC to run for. Default (`ngen = 100000000`) is very high.

`doNotRun` If `TRUE`, the commands that cause a script to automatically begin running in *MrBayes* will be left out. Useful for troubleshooting initial runs of scripts for non-fatal errors and warnings (such as ignored constraints). Default for this argument is `FALSE`.

`autoCloseMrB` If `TRUE`, the MrBayes script created by this function will 'autoclose', so that when an MCMC run finishes the specified number of generations, it does not interactively check whether to continue the MCMC. This is often necessary for batch analyses.

`cleanNames` If `TRUE` (the default), then special characters (currently, this only contains the forward-slashes: '/') are removed from taxon names before construction of the NEXUS file.

`printExecute` If `TRUE` (the default) and if output is directed to a `newFile` (i.e. a `newFile` is specified), a line for pasting into MrBayes for executing the newly created file will be messaged to the terminal.

Details

Users must supply a data set of tip ages (in various formats), which are used to construct age calibrations commands on the tip taxa (via paleotree function `createMrBayesTipCalibrations`). The user must also supply some topological constraint: either a set of taxa designated as the outgroup, which is then converted into a command constraining the monophyly on the ingroup taxa, which is presumed to be all taxa *not* listed in the outgroup. Alternatively, a user may supply a tree which is

then converted into a series of hard topological constraints (via function `createMrBayesConstraints`. Both types of topological constraints cannot be applied.

Many of the options available with `createMrBayesTipCalibrations` are available with this function, allowing users to choose between fixed calibrations or uniform priors that approximate stratigraphic uncertainty. In addition, the user may also supply a path to a text file presumed to be a NEXUS file containing character data formatted for use with *MrBayes*.

The taxa listed in `tipTimes` must match the taxa in `treeConstraints`, if such is supplied. If supplied, the taxa in `outgroupTaxa` must be contained within this same set of taxa. These all must have matches in the set of taxa in `origNexusFile`, if provided and if `parseOriginalNexus` is `TRUE`.

Note that because the same set of taxa must be contained in all inputs, relationships are constrained as 'hard' constraints, rather than 'partial' constraints, which allows some taxa to float across a partially fixed topology. See the documentation for `createMrBayesConstraints`, for more details.

Value

If argument `newFile` is `NULL`, then the text of the generated NEXUS script is output to the console as a series of character strings.

Note

This function allows a user to take an undated phylogenetic tree in R, and a set of age estimates for the taxa on that tree, and produce a posterior sample of dated trees using the MCMCMC in *MrBayes*, while treating an 'empty' morphological matrix as an uninformative set of missing characters. This 'clock-less tip-dating' approach is essentially an alternative to the *cal3* method in paleotree, sharing the same fundamental theoretical model (a version of the fossilized birth-death model), but with a better algorithm that considers the whole tree simultaneously, rather than evaluating each node individually, from the root up to the tips (as *cal3* does it, and which may cause artifacts). That said, *cal3* still has a few advantages: tip-dating as of April 2017 still only treats OTUs as point observations, contained in a single time-point, while cal3 can consider taxa as having durations with first and last occurrences. This means it may be more straightforward to assess the extent of budding cladogenesis patterns of ancestor-descendant relationships in *cal3*, than in tip-dating.

Author(s)

David W. Bapst. This code was produced as part of a project funded by National Science Foundation grant EAR-1147537 to S. J. Carlson.

The basic *MrBayes* commands utilized in the output script are a collection of best practices taken from studying NEXUS files supplied by April Wright, William Gearty, Graham Slater, Davey Wright, and guided by the recommendations of Matzke and Wright, 2016 in Biology Letters.

References

The basic fundamentals of tip-dating, and tip-dating with the fossilized birth-death model are introduced in these two papers:

Ronquist, F., S. Klopfstein, L. Vilhelmsen, S. Schulmeister, D. L. Murray, and A. P. Rasnitsyn. 2012. A Total-Evidence Approach to Dating with Fossils, Applied to the Early Radiation of the Hymenoptera. *Systematic Biology* 61(6):973-999.

Zhang, C., T. Stadler, S. Klopfstein, T. A. Heath, and F. Ronquist. 2016. Total-Evidence Dating under the Fossilized Birth-Death Process. *Systematic Biology* 65(2):228-249.

For recommended best practices in tip-dating analyses, please see:

Matzke, N. J., and A. Wright. 2016. Inferring node dates from tip dates in fossil Canidae: the importance of tree priors. *Biology Letters* 12(8).

The rationale behind the two alternative morphological models are described in more detail here:

Bapst, D. W., H. A. Schreiber, and S. J. Carlson. 2018. Combined Analysis of Extant Rhynchonellida (Brachiopoda) using Morphological and Molecular Data. *Systematic Biology* 67(1):32-48.

See Also

This function wraps various aspects of the functions `createMrBayesConstraints` and the function `createMrBayesTipCalibrations`. In many ways, this functionality is a replacement for the probabilistic dating method represented by the `cal3` dating functions.

For putting the posterior estimated trees on an absolute time scale, see functions `obtainDatedPosteriorTreesMrB`. Use the argument `getFixedTimes = TRUE` if you used a taxon with a fixed age, and function `setRootAges` to set the root age.

Examples

```
# load retiolitid dataset
data(retiolitinae)

# let's try making a NEXUS file!

# Use a uniform prior, with a 10 million year offset for
 # the expected tree age from the earliest first appearance

# Also set average tree age to be 10 Ma earlier than first FAD

outgroupRetio <- "Rotaretiolites"
# this taxon will now be sister to all other included taxa

# the following will create a NEXUS file
  # with an 'empty' morph matrix
 # where the only topological constraint is on ingroup monophyly
 # Probably shouldn't do this: leaves too much to the FBD prior

# with doNotRun set to TRUE for troubleshooting

createMrBayesTipDatingNexus(
tipTimes = retioRanges,
outgroupTaxa = outgroupRetio,
treeConstraints = NULL,
ageCalibrationType = "uniformRange",
whichAppearance = "first",
treeAgeOffset = 10,
newFile = NULL,
origNexusFile = NULL,
```

```
createEmptyMorphMat = TRUE,
runName = "retio_dating",
doNotRun = TRUE
)

# let's try it with a tree for topological constraints
     # this requires setting outgroupTaxa to NULL
# let's also set doNotRun to FALSE

createMrBayesTipDatingNexus(
   tipTimes = retioRanges,
outgroupTaxa = NULL,
treeConstraints = retioTree,
ageCalibrationType = "uniformRange",
whichAppearance = "first",
treeAgeOffset = 10,
newFile = NULL,
origNexusFile = NULL,
createEmptyMorphMat = TRUE,
runName = "retio_dating",
doNotRun = FALSE
)

# the above is essentially cal3 with a better algorithm,
# and no need for a priori rate estimates
# just need a tree and age estimates for the tips!

#####################################################
# some more variations for testing purposes

# no morph matrix supplied or generated
# you'll need to manually append to an existing NEXUS file

createMrBayesTipDatingNexus(
   tipTimes = retioRanges,
outgroupTaxa = NULL,
treeConstraints = retioTree,
ageCalibrationType = "uniformRange",
whichAppearance = "first",
treeAgeOffset = 10,
newFile = NULL,
origNexusFile = NULL,
createEmptyMorphMat = FALSE,
runName = "retio_dating",
doNotRun = TRUE
)

## Not run:

# let's actually try writing an example with topological constraints
# to file and see what happens

# here's my super secret MrBayes directory
```

```
file <- "D:\\dave\\workspace\\mrbayes\\exampleRetio.nex"

createMrBayesTipDatingNexus(
   tipTimes = retioRanges,
outgroupTaxa = NULL,
treeConstraints = retioTree,
ageCalibrationType = "uniformRange",
whichAppearance = "first",
treeAgeOffset = 10,
newFile = file,
origNexusFile = NULL,
createEmptyMorphMat = TRUE,
runName = "retio_dating",
doNotRun = FALSE
)


## End(Not run)
```

`dateNodes` *Absolute Dates for Nodes of a Time-Scaled Phylogeny*

Description

This function returns the ages of nodes (both internal and terminal tips) for a given phylogeny of class `phylo`. Its use is specialized for application to dated trees from `paleotree`, see Details below.

Usage

```
dateNodes(
  tree,
  rootAge = tree$root.time,
  labelDates = FALSE,
  tolerance = 0.001
)
```

Arguments

`tree`	A phylogeny object of class `phylo`. Must have edge lengths!
`rootAge`	The root age of the tree, assumed by default to be equal to the element at `tree$root.time`, which is a standard element for trees dated by the `paleotree` package. If not given by the user and if the `$root.time` element does not exist, then the maximum depth of the tree will be taken as the root age, which implicitly assumes the latest most terminal tip is an extant taxon at the modern day (time = 0). If `rootAge` is so defined that some nodes may occur later than time = 0, this function may return negative dates.

labelDates	If FALSE (the default), the dates returned are labeled with the tip/node numbers as given in tree$edge. If TRUE, they are labeled with the tip labels of every descendant tip, which for terminal tips means a single taxon label, and for internal tips a label that might be very long, composed of multiple tip labels pasted together. Thus, by default, this argument is FALSE.
tolerance	The tolerance within which a node date has to be removed from zero-time (i.e. the modern) to issue a warning that there are 'negative' node dates.

Details

This function is specialized for dated phylogenies, either estimated from empirical data or simulated with functions from paleotree, and thus have a $root.time element. This function will still work without such, but users should see the details for the rootAge argument.

Value

Returns a vector of length Ntip(tree) + Nnode(tree) which contains the dates for all terminal tip nodes and internal nodes for the tree, in that order, as numbered in the tree$edge matrix. These dates are always on a descending scale (i.e. time before present); see help for argument rootAge for how the present time is determined. If rootAge is so defined that some nodes may occur later than time = 0 units before present, this function may (confusingly) return negative dates and a warning message will be issued.

Author(s)

David W. Bapst, based on a function originally written by Graeme Lloyd.

See Also

compareTimescaling, nodeDates2branchLengths

Examples

```
#let's simulate some example data
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
#get the true time-sclaed tree
tree1 <- taxa2phylo(taxa)

#now let's try dateNodes
dateNodes(tree1)

#let's ignore $root.time
dateNodes(tree1,rootAge = NULL)

#with the lengthy tip-label based labels
   #some of these will be hideously long
dateNodes(tree1,labelDates = TRUE)
```

`dateTaxonTreePBDB` *Date a Taxon-Topology from the Paleobiology Database Using Appearance Data from the API*

Description

The function `dateTaxonTreePBDB` takes a input consisting of a topology, with tip and internal node labels corresponding to taxa in the Paleobiology Database, and a table of data (containing those same tip and node taxa) obtained from the taxa-list functionality of the Paleobiology Database's API, with appearance times. This function will then output a tree with nodes reflecting the ages of the respective higher taxa, based on their earliest times of appearance from the Paleobiology Database.

Usage

```
dateTaxonTreePBDB(
  taxaTree,
  taxaDataPBDB = taxaTree$taxaDataPBDB,
  minBranchLen = 0,
  tipTaxonDateUsed = "shallowestAge",
  dropZeroOccurrenceTaxa = TRUE,
  plotTree = FALSE
)
```

Arguments

`taxaTree`	A tree with tip taxon names matching the taxon names in `taxaDataPBDB`. Probably a taxon tree estimated with `makePBDBtaxonTree`.
`taxaDataPBDB`	A data table of taxonomic information obtained using the Paleobiology Database's API for a set of taxa that includes the tip taxa on `taxaTree`, generated with parameter `show = app` so that appearance times are included.
`minBranchLen`	Following dating using the appearance times taken directly from the PBDB for each tip taxon and node, the tree may then be assessed with the minimum branch length algorithm, as applied by `minBranchLength`. If `minBranchLen = 0`, the default, this step is skipped. It may be necessary to set `minBranchLen` higher than zero to differentiate nodes in cases with poor stratigraphic congruency, so that derived taxa are the first taxa observed in a group.
`tipTaxonDateUsed`	Controls what date for a taxon from the PBDB is used for 'when' the tip should be placed in the dated phylogeny produced by this function. The default, `tipTaxonDateUsed = "shallowestAge"` will use the minimum age of the last appearance time of that taxon, which if it is extant will be 0, and if it is extinct, will be the maximum constraint on the age of its last appearance (i.e. the last time we saw it before it went extinct). A second option is `"deepestAge"`, which is the oldest possible first appearance time from the PBDB, i.e. the maximum age constraint for the first appearance. As closely related taxa often first occur in the same

short interval of geologic time, due to diversification bursts and/or the heterogeneity of fossil preservation, this may result in confusing polytomies of many terminal taxa with no terminal branch lengths.

`dropZeroOccurrenceTaxa`
: If `TRUE`, the default, then extinct taxa or extinct clades found to have zero occurrences in the Paleobiology Database are removed. If this option isn't used, the function will likely fail as nodes or tips with `NA` ages listed cannot be processed by `parentChild2TaxonTree`.

`plotTree`
: If `TRUE`, the resulting dated tree is plotted. This is `FALSE` by default.

Details

The dating by this function is very simplistic, representing a rather straight interpretation of what the PBDB reports. The dated trees produced should not be taken overly seriously.

Value

Returns a dated phylogeny of class `phylo`, with an additional element `$taxaDataPBDB` added containing the input `taxaDataPBDB`, as this might be called by other functions.

Author(s)

David W. Bapst

References

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

The equuid tree used in the examples is from: MacFadden, B. J. 1992. Fossil horses: systematics, paleobiology, and evolution of the family Equidae. *Cambridge University Press*.

See Also

See `getDataPBDB`, `makePBDBtaxonTree`, and `plotPhyloPicTree`.

Examples

```
taxaAnimals<-c("Archaeopteryx", "Eldredgeops",
"Corvus", "Acropora", "Velociraptor", "Gorilla",
"Olenellus", "Lingula", "Dunkleosteus",
"Tyrannosaurus", "Triceratops", "Giraffa",
"Megatheriidae", "Aedes", "Histiodella",
"Rhynchotrema", "Pecten", "Homo", "Dimetrodon",
"Nemagraptus", "Panthera", "Anomalocaris")

data <-getSpecificTaxaPBDB(taxaAnimals)
tree <- makePBDBtaxonTree(data, rankTaxon = "genus")
```

```
#get the ranges
timeTree <- dateTaxonTreePBDB(tree)

plotPhyloPicTree(tree = timeTree,
     depthAxisPhylo = TRUE)



####################################

## Not run:

# can also plot dated tree with strap

library(strap)
#now plot it
strap::geoscalePhylo(
    tree=timeTree,
    direction = "upwards",
    ages=rangesMinMax,
    cex.tip=0.7,
    cex.ts=0.55,
    cex.age=0.5,
    width=3,
    tick.scale = 50,
    quat.rm=TRUE,
    boxes = "Period",
    arotate = 90,
    units=c("Eon","Period","Era"),
    x.lim=c(650,-20)
    )

## End(Not run)

#################################################################



# we can also use this for pre-existing trees
    # for example, this tree of equuids (horses)
    # borrowed from UCMP materials on horse evolution
    # https://evolution.berkeley.edu/evolibrary/images/HorseTree.pdf
    # (apparently from MacFadden, 1992? Citation above)

# read the tree in as Newick string
horseTree <- ape::read.tree(file=NULL,
    text = paste0(
         "(Eohippus,(Xenicohippus,(Haplohippus,(Epihippus,",
         "(Miohippus,(((Hypohippus,Megahippus),(Anchitherium,",
         "Kalobatippus)),(Archaeohippus,(Desmatippus,(Parahippus,",
         "(Merychippus,(((Hipparion_Merychippus,(Nannippus,",
         " Cormohipparion)),(Pseudhipparion,(Neohipparion,",
```

```
          " Hipparion))),(Equine_Merychippus,((Protohippus,Calippus),",
          "(Pliohippus,(Astrohippus,(Dinohippus,Equus)))))))))))))))));"
          )
    )

# note there is a message that the tree lacks node names
    # this is unexpected / atypical for taxon trees

plot(horseTree)

# now let's get data on the tip from the PBDB
    # using getSpecificTaxaPBDB
horseData <- getSpecificTaxaPBDB(horseTree$tip.label)

# now we can date the tree with dateTaxonTreePBDB

datedHorseTree <- dateTaxonTreePBDB(
    taxaTree = horseTree,
    taxaDataPBDB = horseData,
    minBranchLen = 1
    )

# and let's try plotting it!
plotPhyloPicTree(
    tree = datedHorseTree,
    depthAxisPhylo = TRUE
    )

# a fairly boring phylopic diagram
    # not many horse phylopics as of 07-16-19?


## Not run:

# Let's look at this horse tree with strap

library(strap)

geoscalePhylo(
    tree = datedHorseTree,
    ages = datedHorseTree$ranges.used,
    cex.tip = 0.7,
    cex.ts = 0.7,
    cex.age = 0.7,
    width = 4,
    tick.scale = 15,
    boxes = "Epoch",
    erotate = 90,
    quat.rm = TRUE,
    units = c("Period","Epoch"),
    x.lim = c(65,-10)
    )
```

```
## End(Not run)
```

---

degradeTree *Randomly Collapse a Portion of Nodes on a Phylogeny*

Description

degradeTree removes a proportion of the total nodes in a tree, chosen randomly, collapsing the nodes to produce a less-resolved tree. The related function collapseNodes given a tree and a vector of nodes to collapse, removes those nodes from a tree, creating a polytomy.

Usage

```
degradeTree(
  tree,
  prop_collapse = NULL,
  nCollapse = NULL,
  node.depth = NA,
  leave.zlb = FALSE
)

collapseNodes(tree, nodeID, collapseType, leave.zlb = FALSE)
```

Arguments

tree	A phylogeny of class phylo
prop_collapse	Proportion of nodes to collapse
nCollapse	Number of nodes to collapse, can be supplied as an alternative to prop_collapse
node.depth	A number between 0 to 1, which conditions the depth of nodes removed. If NA, no conditioning (this is the default).
leave.zlb	If FALSE, the default option, the original branch length distribution is destroyed and branches set to zero by this function will return polytomies. If TRUE, then the original edge lengths are kept for unmodified edges, and modified edges are changed to zero length, and are not collapsed into polytomies. The removed branch length is not shifted to other edges.
nodeID	The node ID number(s) to be collapsed into a polytomy, as identified in the $edge matrix of the phylo object. Must be a vector of one or more ID numbers.
collapseType	Whether to collapse the edge leading the listed node (if collapseType = "forward"), or to collapse the child edges leading away from the node (if collapseType = "backward"). Collapsing a node ' 'into' a polytomy conceptually could be either and users should heed this option carefully. A third option, if "collapseType = clade" is to collapse the entire clade that is descended from a node (i.e. forward).

Details

In the function degradeTree, the nodes are removed at random using the basic R function sample. degradeTree can be conditioned to remove nodes of a particular depth with greater probability/frequency by setting node.depth to a value between zero (favoring the removal of deep nodes close to the root) or one (shallow nodes far from the root). Depth is evaluated based on the number of descendant tips. If node.depth is not NA, the relative proportion of descendants from each node is calculated, summed to 1 and the node.depth value subtracted from this proportion. These values are then squared, normalized again to equal to 1 and then used as the probabilities for sampling nodes for removal.

By default, branch lengths are removed from the input tree prior to degradation and entirely absent from the output tree. This is changed if argument leave.zlb is TRUE.

Value

Returns the modified tree as an object of class phylo, with no edge lengths by default.

Author(s)

David W. Bapst

See Also

di2multi,timeLadderTree

Examples

```
set.seed(444)
tree <- rtree(100)
tree1 <- degradeTree(tree,prop_collapse = 0.5)
tree3 <- degradeTree(tree,nCollapse = 50)

#let's compare the input and output
layout(matrix(1:2,,2))
plot(tree,show.tip.label = FALSE,use.edge.length = FALSE)
plot(tree1,show.tip.label = FALSE,use.edge.length = FALSE)

#now with collapseNodes
tree <- rtree(10)
#collapse nodes backwards
   #let's collapse lucky node number 13!
tree1 <- collapseNodes(nodeID = 13,tree = tree,collapseType = "backward")
#collapse nodes forwards
tree2 <- collapseNodes(nodeID = 13,tree = tree,collapseType = "forward")
#collapse entire clade
tree3 <- collapseNodes(nodeID = 13,tree = tree,collapseType = "clade")

#let's compare
layout(1:4)
plot(tree,use.edge.length = FALSE,main = "original")
plot(tree1,use.edge.length = FALSE,main = "backward collapse")
```

```
plot(tree2,use.edge.length = FALSE,main = "forward collapse")
plot(tree3,use.edge.length = FALSE,main = "entire clade")

layout(1)
```

---

depthRainbow *Paint Tree Branch Depth by Color*

Description

Paints the edges of a phylogeny with colors relative to their depth.

Usage

```
depthRainbow(tree)
```

Arguments

tree	A phylogeny, as an object of class phylo.

Details

The only purpose of this function is to make an aesthetically-pleasing graphic of one's tree, where branches are color-coded with a rainbow palette, relative to their depth. Depth is defined relative to the number of branching nodes between the basal node of a branch and the root, not the absolute distance (i.e. branch length) to the root or the distance from the tips.

Value

No value returned, just plots a colorful phylogeny.

Examples

```
set.seed(444)
tree <- rtree(500)
depthRainbow(tree)
```

divCurveFossilRecordSim *Diversity-Curve Plotting for Simulations of Diversification and Sampling In the Fossil Record*

Description

An extremely simple plotting function, which plots the original taxonomic diversity versus the sampled taxonomic diversity, for use with output from the function `simFossilRecord`. If sampling processes were not included in the model, then it plots simply the single diversity curve.

Usage

```
divCurveFossilRecordSim(
  fossilRecord,
  merge.cryptic = TRUE,
  plotLegend = TRUE,
  legendPosition = "topleft",
  curveColors = c("black", "red"),
  curveLineTypes = c(1, 2)
)
```

Arguments

`fossilRecord`	A list object output by `simFossilRecord`, often composed of multiple elements, each of which is data for 'one taxon', with the first element being a distinctive six-element vector composed of numbers, corresponding to the six variable tables by `fossilRecord2fossilTaxa` after simulating with `simFossilRecord` (originally produced by deprecated function `simFossilTaxa`).
`merge.cryptic`	If `TRUE`, cryptic taxon-units (i.e. those in the same cryptic complex) will be merged into single taxa for the sake of being counted in the diversity curves presented by this function.
`plotLegend`	A logical. Should a legend be plotted? Only applies if sampling processes were modeled.
`legendPosition`	Where should the legend be plotted? See help for `legend` for details. Only applies if sampling processes were modeled.
`curveColors`	A vector of length two indicating what colors the original and sampled diversity curves should be displayed in. Only applies if sampling processes were modeled.
`curveLineTypes`	A vector of length two indicating what colors the original and sampled diversity curves should be displayed in. Only applies if sampling processes were modeled.

Details

This function is essentially a wrapper for `paleotree` function `multiDiv`.

Value

This function returns nothing: it just creates a plot.

Author(s)

David W. Bapst

See Also

simFossilRecord

Examples

```
set.seed(44)
record <- simFossilRecord(p = 0.1, q = 0.1, r = 0.1, nruns = 1,
nTotalTaxa = c(20,30) ,nExtant = 0, plot = FALSE)

# now let's plot it
divCurveFossilRecordSim(record)
```

DiversityCurves *Diversity Curves*

Description

Functions to plot diversity curves based on taxic range data, in both discrete and continuous time, and for phylogenies.

Usage

```
taxicDivCont(
  timeData,
  int.length = 1,
  int.times = NULL,
  plot = TRUE,
  plotLogRich = FALSE,
  timelims = NULL,
  drop.cryptic = FALSE
)

taxicDivDisc(
  timeList,
  int.times = NULL,
  drop.singletons = FALSE,
  plot = TRUE,
  plotLogRich = FALSE,
```

```
  timelims = NULL,
  extant.adjust = 0.001,
  split.int = TRUE
)

phyloDiv(
  tree,
  int.length = 0.1,
  int.times = NULL,
  plot = TRUE,
  plotLogRich = FALSE,
  drop.ZLB = TRUE,
  timelims = NULL
)
```

Arguments

timeData	Two-column matrix giving the per-taxon first and last appearances in absolute time. The simulated data tables output by fossilRecord2fossilTaxa following simulation with simFossilRecord can also be supplied to taxicDivCont.
int.length	The length of intervals used to make the diversity curve. Ignored if int.times is given.
int.times	An optional two-column matrix of the interval start and end times for calculating the diversity curve. If NULL, calculated internally. If given, the argument split.int and int.length are ignored.
plot	If TRUE, a diversity curve generated from the data is plotted.
plotLogRich	If TRUE, taxic diversity is plotted on log scale.
timelims	Limits for the x (time) axis for diversity curve plots. Only affects plotting. Given as either NULL (the default) or as a vector of length two as for xlim in the basic R function plot. Time axes will be plotted *exactly* to these values.
drop.cryptic	If TRUE, cryptic taxa are merged to form one taxon for estimating taxon curves. Only works for objects from simFossilRecord via fossilRecord2fossilTaxa.
timeList	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. See details.
drop.singletons	If TRUE, taxa confined to a single interval will be dropped prior to the diversity curve calculation. This is sometimes done if single intervals have overly high diversities due to the 'monograph' effect where more named taxa are known in certain intervals largely due to taxonomic expert effort and not real changes in historical biotic diversity.
extant.adjust	Amount of time to be added to extend start time for (0,0) bins for extant taxa, so that the that 'time interval' does not appear to have an infinitely small width.
split.int	For discrete time data, should calculated/input intervals be split at discrete time interval boundaries? If FALSE, can create apparent artifacts in calculating the diversity curve. See details.
tree	A time-scaled phylogeny of class phylo.

drop.ZLB — If TRUE, zero-length terminal branches are dropped from the input tree for phylogenetic datasets, before calculating standing diversity.

Details

First, some background. Diversity curves are plots of species/taxon/lineage richness over time for a particular group of organisms. For paleontological studies, these are generally based on per-taxon range data while more recently in evolutionary biology, molecular phylogenies have been used to calculate lineage-through-time plots (LTTs). Neither of these approaches are without their particular weaknesses; reconstructing the true history of biodiversity is a difficult task no matter what data is available.

The diversity curves produced by these functions will always measure diversity within binned time intervals (and plot them as rectangular bins). For continuous-time data or phylogenies, one could decrease the int.length used to get what is essentially an 'instantaneous' estimate of diversity. This is warned against, however, as most historical diversity data will have some time-averaging or uncertain temporal resolution and thus is probably not finely-resolved enough to calculate instantaneous estimates of diversity.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

As diversity is counted within binned intervals, the true standing diversity may be somewhat lower than the measured (observed) quantity, particularly if intervals are longer than the mean duration of taxa is used. This will be an issue with all diversity curve functions, but particularly the discrete-time variant. For diversity data in particularly large discrete time intervals, plotting this data in smaller bins which do not line up completely with the original intervals will create a 'spiky' diversity curve, as these smaller intersecting bins will have a large number of taxa which may have been present in either of the neighboring intervals. This will give these small bins an apparently high estimated standing diversity. This artifact is avoided with the default setting split.int = TRUE, which will split any input or calculated intervals so that they start and end at the boundaries of the discrete-time range bins.

The timeList object should be a list composed of two matrices, the first matrix giving by-interval start and end times (in absolute time), the second matrix giving the by-taxon first and last appearances in the intervals defined in the first matrix, numbered as the rows. Absolute time should be decreasing, while the intervals should be numbered so that the number increases with time. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. See the documentation for the time-scaling function bin_timePaleoPhy and the simulation function binTimeData for more information on formatting.

Unlike some paleotree functions, such as perCapitaRates, the intervals can be overlapping or of unequal length. The diversity curve functions deal with such issues by assuming taxa occur from the base of the interval they are first found in until the end of the last interval they are occur in. Taxa in wide-ranging intervals that contain many others will be treated as occurring in all nested intervals.

phyloDiv will resolve polytomies to be dichotomous nodes separated by zero-length branches prior to calculating the diversity curve. There is no option to alter this behavior, but it should not affect the use of the function because the addition of the zero-length branches should produce an identical diversity history as a polytomy. phyloDiv will also drop zero-length terminal branches, as with the function dropZLB. This the default behavior for the function but can be turned off by setting the argument drop.zlb to FALSE.

Value

These functions will invisibly return a three-column matrix, where the first two columns are interval start and end times and the third column is the number of taxa (or lineages) counted in that interval.

Author(s)

David W. Bapst

See Also

multiDiv, timeSliceTree, binTimeData

There are several different functions for traditional LTT plots (phylogenetic diversity curves), such as the function ,ltt.plot in the package ape, the function ltt in the package phytools, the function plotLtt in the package laser and the function LTT.average.root in the package TreeSim.

Examples

```
# taxicDivDisc example with the retiolinae dataset
data(retiolitinae)
taxicDivDisc(retioRanges)

################################################

# simulation examples

# 07-15-19
# note that the examples below are weird and rather old
    # the incomplete sampling can now be done
    # with the same function that simulates diversification

set.seed(444)

record <- simFossilRecord(
       p = 0.1,
       q = 0.1,
       nruns = 1,
       nTotalTaxa = c(30,40),
       nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)

# let's see what the 'true' diversity curve looks like in this case
#plot the FADs and LADs with taxicDivCont
taxicDivCont(taxa)

# simulate a fossil record with imperfect sampling via sampleRanges
rangesCont <- sampleRanges(taxa, r = 0.5)

# plot the diversity curve based on the sampled ranges
layout(1:2)
taxicDivCont(rangesCont)
```

```
# Now let's use binTimeData to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,
       int.length = 1)
# plot with taxicDivDisc
taxicDivDisc(rangesDisc)
# compare to the continuous time diversity curve

layout(1)
# Now let's make a tree using taxa2phylo
tree <- taxa2phylo(taxa,obs_time = rangesCont[,2])
phyloDiv(tree)

# a simple example with phyloDiv
  # using a tree from rtree in ape
set.seed(444)
tree <- rtree(100)
phyloDiv(tree)

############################################################

#a neat example of using phyDiv with timeSliceTree
 #to simulate doing molecular-phylogeny studies
 #of diversification...in the past

set.seed(444)
record <- simFossilRecord(
       p = 0.1,
       q = 0.1,
       nruns = 1,
       nTotalTaxa = c(30,40),
       nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
taxicDivCont(taxa)

#that's the whole diversity curve

#with timeSliceTree we could look at the lineage accumulation curve
 #we'd get of species sampled at a point in time

tree <- taxa2phylo(taxa)
#use timeSliceTree to make tree of relationships up until time = 950
tree950 <- timeSliceTree(tree,
       sliceTime = 950,
       plot = TRUE,
       drop.extinct = FALSE)

#use drop.extinct = TRUE to only get the tree of lineages extant at time = 950
tree950 <- timeSliceTree(tree,
       sliceTime = 950,
       plot = TRUE,
       drop.extinct = TRUE)

#now its an ultrametric tree with many fewer tips...
```

```
#lets plot the lineage accumulation plot on a log scale
phyloDiv(tree950,
       plotLogRich = TRUE)

################################################
#an example of a 'spiky' diversity curve
       # and why split.int is a good thing
set.seed(444)
record <- simFossilRecord(
       p = 0.1,
       q = 0.1,
       nruns = 1,
       nTotalTaxa = c(30,40),
       nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)

taxaDiv <- taxicDivCont(taxa)

#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa, r = 0.5)
rangesDisc <- binTimeData(rangesCont,
       int.length = 10)

#now let's plot with taxicDivDisc
 # but with the intervals from taxaDiv
 # by default, split.int = TRUE

taxicDivDisc(rangesDisc,
       int.times = taxaDiv[,1:2],
       split.int = TRUE)

#look pretty!

#now let's turn off split.int
taxicDivDisc(rangesDisc,
       int.times = taxaDiv[,1:2],
       split.int = FALSE)
#looks 'spiky'!
```

`durationFreq` *Models of Sampling and Extinction for Taxonomic Duration Datasets*

Description

These functions construct likelihood models of the observed frequency of taxon durations, given either in discrete (`make_durationFreqDisc`) or continuous time (`make_durationFreqCont`). These models can then be constrained using functions available in this package and/or analyzed with commonly used optimizing functions.

Usage

```
make_durationFreqCont(
  timeData,
  groups = NULL,
  drop.extant = TRUE,
  threshold = 0.01,
  tol = 1e-04
)

make_durationFreqDisc(timeList, groups = NULL, drop.extant = TRUE)
```

Arguments

timeData	Two-column matrix of per-taxon first and last occurrence given in continuous time, relative to the modern (i.e. older dates are also the 'larger' dates). Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs the supplied matrix) are automatically dropped from the matrix and from groups simultaneously.
groups	Either NULL (the default) or matrix with the number of rows equal to the number of taxa and the number of columns equal to the number of 'systems' of categories for taxa. Taxonomic membership in different groups is indicated by numeric values. For example, a dataset could have a 'groups' matrix composed of a column representing thin and thick shelled taxa, coded 1 and 2 respectively, while the second column indicates whether taxa live in coastal, outer continental shelf, or deep marine settings, coded 1-3 respectively. Different combinations of groups will be treated as having independent sampling and extinction parameters in the default analysis, for example, thinly-shelled deep marine species will have separate parameters from thinly-shelled coastal species. Grouping systems could also represent temporal heterogeneity, for example, categorizing Paleozoic versus Mesozoic taxa. If groups are NULL (the default), all taxa are assumed to be of the same group with the same parameters. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in timeData or timeList) are automatically dropped from groupings and the time dataset (either timeData or timeList) and from groups simultaneously.
drop.extant	Drops all extant taxa from a dataset before preceding.
threshold	The smallest allowable duration (i.e. the measured difference in the first and last occurrence dates for a given taxon). Durations below this size will be treated as "one-hit" sampling events.
tol	Tolerance level for determining whether a taxon from a continuous-time analysis is extant or not. Taxa which occur at a date less than tol are treated as occurring at the modern day (i.e. being functionally identical as occurring at 0 time).
timeList	A two column matrix, with the first and last occurrences of taxa given in relative time intervals (i.e. ordered from first to last). If a list of length = 2 is given for timeData, such as would be expected if the output of binTimeData was used as the input, the second element is used. See details. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in the second matrix) are automatically dropped from the timeList and from groups simultaneously.

Living taxa observed in the modern day are expected to be listed as last observed in a special interval (`c(0,0)`), i.e. begins and ends at zero (modern) time. This interval is always automatically removed prior to the calculation intermediary data for fitting likelihood functions.

Details

These functions effectively replace two older functions in paleotree, now removed, `getSampRateCont` and `getSampProbDisc`. The functions here do not offer the floating time interval options of their older siblings, but do allow for greater flexibility in defining constrains on parameter values. Differences in time intervals, or any other conceivable discrete differences in parameters, can be modeled using the generic `groups` argument in these functions.

These functions use likelihood functions presented by Foote (1997). These analyses are ideally applied to data from single stratigraphic section but potentially are applicable to regional or global datasets, although the behavior of those datasets is less well understood.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero and older dates are 'larger'. On the contrary, relative time is in intervals with non-zero integers that increase sequentially beginning with 1, from earliest to oldest.

For `make_durationFreqDisc`, the intervals in `timeList` should be non-overlapping sequential intervals of roughly equal length. These should be in relative time as described above, so the earliest interval should be listed as `1` and the numbering should increase as the intervals go up with age. If both previous statements are `TRUE`, then differences in interval numbers will represent the same rough difference in the absolute timing of those intervals. For example, a dataset where all taxa are listed from a set of sequential intervals of similar length, such as North American Mammal assemblage zones, microfossil faunal zones or graptolite biozones can be given as long as they are correctly numbered in sequential order in the input. As a counter example, a dataset which includes taxa resolved only to intervals as wide as the whole Jurassic and taxa resolved to biozones within the Jurassic should not be included in the same input. Drop taxa from less poorly resolved intervals from such datasets if you want to apply this function, as long as this retains a large enough sample of taxa listed from the sequential set of intervals.

Please check that the optimizer function you select actually converges. The likelihood surface can be very flat in some cases, particularly for small datasets (<100 taxa). If the optimizer does not converge, consider increasing iterations or changing the starting values.

Value

A function of class `"paleotreeFunc"`, which takes a vector equal to the number of parameters and returns the *negative* log-likelihood (for use with `optim` and similar optimizing functions, which attempt to minimize support values). See the functions listed at `modelMethods` for manipulating and examining such functions and `constrainParPaleo` for constraining parameters.

Parameters in the output functions are named `q` for the instantaneous per-capita extinction rate, `r` for the instantaneous per-capita sampling rate and `R` for the per-interval taxonomic sampling probability. Groupings follow the parameter names, separated by periods; by default, the parameters will be placed in at least group '`1`' (of a grouping scheme containing only a single group), such that `make_durationFreqCont` by default creates a function with parameters named `q.1` and `r.1`, while `make_durationFreqDisc` creates a function with parameters named `q.1` and `R.1`.

Note that the q parameters estimated by make_durationFreqDisc is scaled to per lineage intervals and not to per lineage time-units. If intervals are the same length, this can be easily corrected by multiplying one by the interval length. It is unclear how to treat uneven intervals and I urge users to consider multiple strategies.

For translating these sampling probabilities and sampling rates, see SamplingConv.

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

See Also

See freqRat, sRate2sProb, qsRate2Comp sProb2sRate and qsProb2Comp.

Examples

```
# let's simulate some taxon ranges from
   # an imperfectly sampled fossil record
set.seed(444)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
   nTotalTaxa = c(30,40),
   nExtant = 0
   )
taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r = 0.5)
#bin the ranges into discrete time intervals
rangesDisc <- binTimeData(rangesCont,int.length = 1)
#note that we made interval lengths = 1:
  # thus q (per int) = q (per time) for make_durationFreqDisc

## Not run:
#old ways of doing it (defunct as of paleotree version 2.6)
getSampRateCont(rangesCont)
getSampProbDisc(rangesDisc)

## End(Not run)

#new ways of doing it
    # we can constrain our functions
    # we can use parInit, parLower and parUpper
    # to control parameter bounds
```

```
#as opposed to getSampRateCont, we can do:
likFun <- make_durationFreqCont(rangesCont)
optim(parInit(likFun),
      likFun,
      lower = parLower(likFun),
      upper = parUpper(likFun),
      method = "L-BFGS-B",
      control = list(maxit = 1000000)
      )

#as opposed to getSampProbDisc, we can do:

likFun <- make_durationFreqDisc(rangesDisc)
optim(parInit(likFun),
      likFun,
      lower = parLower(likFun),
      upper = parUpper(likFun),
      method = "L-BFGS-B",
      control = list(maxit = 1000000)
      )

#these give the same answers (as we'd expect them to!)

#with newer functions we can constrain our functions easily
    # what if we knew the extinction rate = 0.1 a priori?

likFun <- make_durationFreqCont(rangesCont)
likFun <- constrainParPaleo(likFun,q.1~0.1)
optim(parInit(likFun),
      likFun,
      lower = parLower(likFun),
      upper = parUpper(likFun),
     method = "L-BFGS-B",
     control = list(maxit = 1000000)
     )

#actually decreases our sampling rate estimate
   # gets further away from true generating value, r = 0.5 (geesh!)
   # but this *is* a small dataset...
```

`equation2function` *Turn a Character String of the Right-Hand Side of an Equation into an R Function*

Description

`equation2function` converts the right-hand side of an equation that can be written as a single line (like the right-hand side of an object of class `formula`) and creates an R function which calls the variables within as arguments and returns values consistent with the parameters of the input equation as written.

Usage

```
equation2function(equation, envir = parent.frame(), notName = "XXXXXXXXXXX")
```

Arguments

equation	The right-hand-side (RHS) of an equation, given as a character string. If not of type character, equation2function attempts to coerce equation to type character and fails if it cannot.
envir	The environment the resulting function will be evaluated in. See as.function.
notName	A useless string used simply as a placeholder in turning equation into a function, which should not match any actual variable in equation. Only supplied as an argument in case any

Details

This simple little function is rather 'hacky' but seems to get the job done, for a functionality that does not seem to be otherwise exist elsewhere in R.

Value

A function, with named blank (i.e. no default value) arguments.

Author(s)

David W. Bapst

Examples

```
# some simple examples
foo <- equation2function("x+y")
foo
foo(x = 4,y = 0.1)

foo <- equation2function("x+2*sqrt(2*y+3)^2")
foo
foo(x = 4,y = 0.1)

# what about weird long argument names and spaces
foo <- equation2function("stegosaur + 0.4 * P")
foo
foo(stegosaur = 5,P = 0.3)
```

exhaustionFunctions *Analyses of the exhaustion of Character States Over Evolutionary History*

Description

The following functions are for measuring and fitting various distributions to the gradual exhaustion of unexpressed character states, as originally described by Wagner (2000, Evolution).

Usage

```
accioExhaustionCurve(
  phyloTree,
  charData,
  charTypes = "unordered",
  outgroup = NULL,
  firstAppearances = NULL,
  missingCharValue = "?",
  inapplicableValue = "-"
)

accioBestAcquisitionModel(
  exhaustion_info,
  changesType,
  models = c("exponential", "gamma", "lognormal", "zipf")
)

charExhaustPlot(
  exhaustion_info,
  changesType,
  xlab = "Total Characters",
  ylab = NULL,
  main = NULL,
  xsize = 3
)
```

Arguments

phyloTree	A phylogenetic tree of class phylo as used by package ape.
charData	A data.frame of morphological character codings (a morphological 'matrix'), with taxon units as rows and characters as columns.
charTypes	A vector of length equal to the number of characters in charData, with elements indicating whether the corresponding character in charData is "unordered" or "ordered". However, if length(charTypes) = 1, then it is repeated for all taxa. The default value for this argument is "unordered".
outgroup	A string matching to one of the tip labels as given by tip.label,

`firstAppearances`
: A vector, with length equal to the same number of taxa (rows) as in `charData`, in the same corresponding order.

`missingCharValue`
: The string value indicating a missing character coding value, by default `"?"`.

`inapplicableValue`
: The string value indicating an inapplicable character coding value, by default `"-"`.

`exhaustion_info`
: The list of results output from function `accioExhaustionCurve`.

`changesType`
: A single character value, indicating the character change data to be assessed from the result of the character exhaustion analysis, must be one of either `'totalAcc'` (to the total number of accumulated character changes, ideal for modeling the size and distribution of *state* space) or `'charAlt'` (to plot the total number of character alterations, ideal for modeling the size and distribution of *character* space).

`models`
: A vector of type `character` naming various models to be fit. The default fits the models `"exponential"`, `"gamma"`, `"lognormal"`, and `"zipf"`.

`xlab`
: Label for the X axis; `"Total Characters"` by default.

`ylab`
: Label for the Y axis. If not provided by the user, a label based on the `changesType` argument will be used.

`main`
: Main title label for the plot. If not provided by the user, a label based on the `changesType` argument will be used.

`xsize`
: Parameter controlling size of the axes, which are forced to be symmetric.

Details

`accioExhaustionCurve` uses a Sankoff parsimony ancestral-reconstruction algorithm (written by P.J. Wagner, *not* the one from `phangorn` used elsewhere in `paleotree`) to calculate character changes across each branch (internode edge) of a tree, and then reports the counts of character state

`accioBestAcquisitionModel` takes output from `accioExhaustionCurve`, calculates one of two character change indices, and then fits a series of user-selected models to the dataset, returning details pertaining to the best-fit model.

`charExhaustPlot` is a wrapper for `accioBestAcquisitionModel` that produces a plot of the observed character change data against the expectation under the best-fit model.

The functions `accioBestAcquisitionModel` and `charExhaustPlot` offer users two different options for examining character change: `totalAcc` fits models to the total accumulated number of state changes over the phylogeny, thus using exhaustion to explore the size and distribution of character space. The other option `charAlt` fits models to the number of character that alter from primitive to derived over phylogeny, thus reflecting the size and distribution of state space.

`accioExhaustionCurve` can order its reconstruction of change by stratigraphic order of first appearances. It is unclear what this means.

Value

`accioExhaustionCurve` outputs a list containing two objects: first, a matrix named `exhaustion` consisting of three columns: `"Steps"`, `"Novel_States"`, and `"Novel_Characters"`, respectively giving the counts of these respective values for each branch (internode edge). The second element of this list is named `State_Derivations` and is a count of how often each state, across all characters, was derived relative to the primitive position along each internode edge.

The output of `accioBestAcquisitionModel` is a list object containing information on the best-fit model, the parameters of that model, and the calculated probability distribution function for that model at the same intervals (for use in quantile plots).

`charExhaustPlot` produces a plot, and outputs no data.

Note

This family of functions presented here were originally written by Peter J. Wagner, and then modified and adapted by David W. Bapst for wider release in a CRAN-distributed package: `paleotree`. This makes the code presented here a very different beast than typical `paleotree` code, for example, there are fewer tests of correct input type, length, etc.

Author(s)

Initially written by Peter J. Wagner, with modification and documentation by David W. Bapst.

References

Wagner, P. J. 2000. Exhaustion of morphologic character states among fossil taxa. *Evolution* 54(2):365-386.

See Also

Also see `paleotree` functions `minCharChange` and `ancPropStateMat`, the latter of which is a wrapper for `phangorn`'s function `ancestral.pars`.

Examples

```
# get data
data(SongZhangDicrano)

dicranoTree <- cladogramDicranoX13

# modify char data
charMat <- data.matrix(charMatDicrano)
charMat[is.na(charMatDicrano)] <- 0
charMat <- (charMat-1)
colnames(charMat) <- NULL
# replace missing values
charMat[is.na(charMatDicrano)] <- "?"

# the 'outgroup' is Exigraptus
```

```
   # also the first taxon listed in the matrix
exhaustionResults <- accioExhaustionCurve(
   phyloTree = dicranoTree,
   charData = charMat, charTypes = "unordered",
   outgroup = "Exigraptus_uniformis")

# fits models to exhaustion for total accumulation
accioBestAcquisitionModel(
   exhaustion_info = exhaustionResults,
   changesType = "totalAcc",
   models = c("exponential","gamma","lognormal","zipf"))

# plot of exhaustion of total accumulation of character states
charExhaustPlot(exhaustion_info = exhaustionResults,
   changesType = "totalAcc")

# plot of exhaustion of character alterations
charExhaustPlot(exhaustion_info = exhaustionResults,
   changesType = "charAlt")
```

`expandTaxonTree` *Extrapolating Lower-Level Taxon Phylogenies from Higher-Level Taxon Trees*

Description

This function takes a tree composed of higher-level taxa and a vector of lower-level taxa belonging to the set of higher-level taxa included in the input tree and produces a tree composed of the lower-level taxa, by treating the higher-level taxa as unresolved monophyletic polytomies. A user can also mark higher taxa as paraphyletic such that these are secondarily collapsed and do not form monophyletic clades in the output tree.

Usage

```
expandTaxonTree(
  taxonTree,
  taxaData,
  collapse = NULL,
  keepBrLen = FALSE,
  plot = FALSE
)
```

Arguments

`taxonTree`	A phylogeny as an object of class `phylo`, where tips represent some 'higher taxa' that is to be expanded at a lower taxonomic level.

taxaData	Character vector of higher taxa, with elements names equal to the lower taxa. See below.
collapse	Character vector containing names of non-monophyletic higher taxa to be collapsed.
keepBrLen	Logical, decides if branch lengths should be kept or discarded. FALSE by default. See details below.
plot	If TRUE, plots a comparison between input and output trees

Details

The output tree will probably be a rough unresolved view of the relationships among the taxa, due to the treatment of higher-level taxa as polytomies. This is similar to the methods used in Webb and Donoghue (2005) and Friedman (2009). Any analyses should be done by resolving this tree with multi2di in the ape package or via the various time-scaling functions found in this package (paleotree).

The taxaData vector should have one element per lower-level taxon that is to be added to the tree. The name of each element in the vector should be the names of the lower-level taxa, which will be used as new tip labels of the output lower-taxon tree. There should be no empty elements! Otherwise, expandTaxonTree won't know what to do with taxa that aren't being expanded.

By default, all higher-level taxa are treated as monophyletic clades if not otherwise specified. The collapse vector can (and probably should) be used if there is doubt about the monophyly of any higher-level taxa included in the input taxon-tree, so that such a group would be treated as a paraphyletic group in the output tree.

Also by default, the output tree will lack branch lengths and thus will not be dated, even if the input phylogeny is dated. If keepBrLen = TRUE, then the tree's edge lengths are kept and new taxa are added as zero length branches attaching to a node that represents the previous higher-taxon. This tree is probably not useful for most applications, and may even strongly bias some analyses. *USE WITH CAUTION!* The collapse argument, given as a vector, will cause such edges to be replaced by zero-length branches rather than fully collapsing them, which could have odd effects. If collapse is not NULL and keepBrLen = TRUE, a warning is issued that the output probably won't make much sense at all.

Value

Outputs the modified tree as an object of class phylo, with the higher-level taxa expanded into polytomies and the lower-level taxa as the tip labels.

Author(s)

David W. Bapst

References

Friedman, M. 2009 Ecomorphological selectivity among marine teleost fishes during the end-Cretaceous extinction. *Proceedings of the National Academy of Sciences* **106**(13):5218–5223.

Webb, C. O., and M. J. Donoghue. 2005 Phylomatic: tree assembly for applied phylogenetics. *Molecular Ecology Notes* **5**(1):181–183.

See Also

multi2di, bind.tree

Examples

```
set.seed(444)
# lets make our hypothetical simulated tree of higher taxa
taxtr <- rtree(10)
# taxa to place within higher taxa
taxd <- sample(taxtr$tip.label, 30, replace = TRUE)
names(taxd) <- paste(taxd,"_x", 1:30, sep = "")
coll <- sample(taxtr$tip.label,3) #what to collapse?
expandTaxonTree(taxonTree = taxtr, taxaData = taxd,
                collapse = coll, plot = TRUE)
```

---

fixRootTime — *Modify, Drop or Bind Terminal Branches of Various Types (Mainly for Paleontological Phylogenies)*

Description

Modifying a dated tree with $root.time elements often changes the actual timing of the root relative to the tips, such as when dropping tips, extending branches, or shift node ages backwards. When such modifications occur, the function fixRootTime can be used to find the correct root age. This function is mainly used as a utility function called by other tree-modifying functions discussed in the manual page for modifyTerminalBranches. This is typically performed via the function fixRootTime.

Usage

```
fixRootTime(
  treeOrig,
  treeNew,
  testConsistentDepth = TRUE,
  fixingMethod = "matchCladeTransferNodeAge"
)
```

Arguments

treeOrig	A phylo object of a time-scaled phylogeny with a $root.time element.
treeNew	A phylo object containing a modified form of treeOrig (with no extra tip taxa added, but possibly with some tip taxa removed).
testConsistentDepth	A logical, either TRUE or FALSE. If TRUE (the default) the tree's root-to-furthest-tip depth is tested to make sure this depth is not greater than the new $root.time appended to the output tree.

`fixingMethod` must be an character value, with a length of 1.

The default option `fixingMethod = "matchCladeTransferNodeAge"`, will determine the `$root.time` of the new tree by comparing the clades of taxa between the two input trees. The new root age assigned is the age of (*1*) the `treeOrig` clade that contains *all* taxa present in `treeNew` and, if the set of (1) contains multiple clades, (*2*) the clade in the first set that contains the fewest taxa not in `treeNew`.

If `fixingMethod = "rescaleUsingTipToRootDist"`, the `root.time` assigned to `treeNew` is the `$root.time` of `treeOrig`, adjusted based on the change in total tree depth between `treeOrig` and `treeNew`, as measured between the root and the first matching taxon in both trees. The `"rescaleUsingTipToRootDist"` option was the default for `fixRootTime` prior to paleotree v2.3, and is the option used by function `minBranchLength` when applied to a tree with pre-existing root age element.

Value

Gives back a modified phylogeny as a `phylo` object, with a modified `$root.time` element.

Author(s)

David W. Bapst

See Also

`modifyTerminalBranches`, `minBranchLength`

Examples

```
#testing dropPaleoTip... and fixRootTime by extension

#simple example
tree <- read.tree(text = "(A:3,(B:2,(C:5,D:3):2):3);")
tree$root.time <- 10
plot(tree,no.margin = FALSE)
axisPhylo()

# now a series of tests, dropping various tips
(test <- dropPaleoTip(tree,"A")$root.time) #  = 7
(test[2] <- dropPaleoTip(tree,"B")$root.time) #  = 10
(test[3] <- dropPaleoTip(tree,"C")$root.time) #  = 10
(test[4] <- dropPaleoTip(tree,"D")$root.time) #  = 10
(test[5] <- dropPaleoTip(tree,c("A","B"))$root.time) #  = 5
(test[6] <- dropPaleoTip(tree,c("B","C"))$root.time) #  = 10
(test[7] <- dropPaleoTip(tree,c("A","C"))$root.time) #  = 7
(test[8] <- dropPaleoTip(tree,c("A","D"))$root.time) #  = 7

# is it all good? if not, fail so paleotree fails...
```

```
if(!identical(test,c(7,10,10,10,5,10,7,7))){stop("fixRootTime fails!")}
```

`footeValues` *Calculates Values for Foote's Inverse Survivorship Analyses*

Description

This function calculates the intermediary values needed for fitting Foote's inverse survivorship analyses, as listed in the table of equations in Foote (2003), with the analyses themselves described further in Foote (2001) and Foote (2005).

Usage

```
footeValues(p, q, r, PA_n = 0, PB_1 = 0, p_cont = TRUE, q_cont = TRUE, Nb = 1)
```

Arguments

`p`	Instantaneous origination/branching rate of taxa. Under a continuous model, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Under a pulsed mode (`p_cont = FALSE`), `p` is a per-interval 'rate' which can exceed 1 (because diversity can more than double; Foote, 2003a). Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as `q` and `r`.
`q`	Instantaneous extinction rate of taxa. Under a continuous model, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Under a pulsed mode (`q_cont = FALSE`), `q` is a per-interval 'rate' but which cannot be observed to exceed 1 (because you can't have more taxa go extinct than exist). Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as `p` and `r`.
`r`	Instantaneous sampling rate of taxa, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as `p` and `q`.
`PA_n`	The probability of sampling a taxon after the last interval included in a survivorship study. Usually zero for extinct groups, although more logically has the value of 1 when there are still extant taxa (i.e., if the last interval is the Holocene and the group is still alive, the probability of sampling them later is probably 1...). Should be a value between 0 and 1.
`PB_1`	The probability of sampling a taxon before the first interval included in a survivorship study. Should be a value between 0 and 1.

p_cont	If TRUE (the default), then origination is assumed to be a continuous time process with an instantaneous rate. If FALSE, the origination is treated as a pulsed discrete-time process with a probability.
q_cont	If TRUE (the default), then extinction is assumed to be a continuous time process with an instantaneous rate. If FALSE, the extinction is treated as a pulsed discrete-time process with a probability.
Nb	The number of taxa that enter an interval (the b is for 'bottom'). This is an arbitrary constant used to scale other values in these calculations and can be safely set to 1.

Details

Although most calculations in this function agree with the errata for Foote's 2003 table (see references), there were some additional corrections for the probability of D given FL (Prob(D|FL)) made as part of a personal communication in 2013 between the package author and Michael Foote.

Value

Returns a matrix with number of rows equal to the number of intervals (i.e. the length of p, q and r) and named columns representing the different values calculated by the function: "Nb", "Nbt", "NbL", "NFt", "NFL", "PD_bt", "PD_bL", "PD_Ft", "PD_FL", "PA", "PB", "Xbt", "XbL", "XFt" and "XFL".

Author(s)

David W. Bapst, with advice from Michael Foote.

References

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Foote, M. 2003a. Origination and Extinction through the Phanerozoic: A New Approach. *The Journal of Geology* 111(2):125-148.

Foote, M. 2003b. Erratum: Origination and Extinction through the Phanerozoic: a New Approach. *The Journal of Geology* 111(6):752-753.

Foote, M. 2005. Pulsed origination and extinction in the marine realm. *Paleobiology* 31(1):6-20.

Examples

```
#very simple example with three intervals, same value for all parameters

# example rates (for the most part)
rate <- rep(0.1, 3)

#all continuous
footeValues(rate,rate,rate)

# origination pulsed
footeValues(rate,rate,rate,p_cont = FALSE)
```

```
# extinction pulsed
footeValues(rate,rate,rate,q_cont = FALSE)

# all pulsed
footeValues(rate,rate,rate,p_cont = FALSE,q_cont = FALSE)
```

`freqRat` *Frequency Ratio Method for Estimating Sampling Probability*

Description

Estimate per-interval sampling probability in the fossil record from a set of discrete-interval taxon ranges using the frequency-ratio method described by Foote and Raup (1996). Can also calculate extinction rate per interval from the same data distribution.

Usage

```
freqRat(timeData, calcExtinction = FALSE, plot = FALSE)
```

Arguments

`timeData`	A 2 column matrix with the first and last occurrences of taxa given in relative time intervals. If a list of length two is given for `timeData`, such as would be expected if the output of `binTimeData` was directly input, the second element is used.
`calcExtinction`	If `TRUE`, the per-interval, per-lineage extinction rate is estimated as the negative slope of the log frequencies, ignoring single hits (as described in Foote and Raup, 1996.)
`plot`	If `TRUE`, the histogram of observed taxon ranges is plotted, with frequencies on a linear scale

Details

This function uses the frequency ratio ("freqRat") method of Foote and Raup (1996) to estimate the per-interval sampling rate for a set of taxa. This method assumes that intervals are of fairly similar length and that taxonomic extinction and sampling works similar to homogenous Poisson processes. These analyses are ideally applied to data from single stratigraphic section but potentially are applicable to regional or global datasets, although the behavior of those datasets is less well understood.

The frequency ratio is a simple relationship between the number of taxa observed only in a single time interval (also known as singletons), the number of taxa observed only in two time intervals and the number of taxa observed in three time intervals. These respective frequencies, respectively f1, f2 and f3 can then be related to the per-interval sampling probability with the following expression.

$Sampling.Probability = (f2^2)/(f1 * f3)$

This frequency ratio is generally referred to as the 'freqRat' in paleobiological literature.

It is relatively easy to visually test if range data fits expectation that true taxon durations are exponentially distributed by plotting the frequencies of the observed ranges on a log scale: data beyond the 'singletons' category should have a linear slope, implying that durations were originally exponentially distributed. (Note, a linear scale is used for the plotting diagram made by this function when 'plot' is TRUE, so that plot cannot be used for this purpose.)

The accuracy of this method tends to be poor at small interval length and even relatively large sample sizes. A portion at the bottom of the examples in the help file examine this issue in greater detail with simulations. This package author recommends using the ML method developed in Foote (1997) instead, which is usable via the function make_durationFreqDisc.

As extant taxa should not be included in a freqRat calculation, any taxa listed as being in a bin with start time 0 and end time 0 (and thus being extant without question) are dropped before the model fitting it performed.

Value

This function returns the per-interval sampling probability as the "freqRat", and estimates

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

See Also

Model fitting methods in make_durationFreqDisc and make_durationFreqCont. Also see conversion methods in sProb2sRate, qsProb2Comp

Examples

```
# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          nruns = 1,
                         nTotalTaxa = c(30,40),
                         nExtant = 0
                         )
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.1)
# Now let's use binTimeData to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length = 5)
```

```
# now, get an estimate of the sampling rate (we set it to 0.5 above)

# for discrete data we can estimate the sampling probability per interval (R)
    # i.e. this is not the same thing as the instantaneous sampling rate (r)

# can use sRate2sProb to see what we would expect
sRate2sProb(r = 0.1, int.length = 5)

# expect R = ~0.39

# now we can apply freqRat to get sampling probability
SampProb <- freqRat(rangesDisc, plot = TRUE)
SampProb

# I estimated R = ~0.25
# Not wildly accurate, is it?

# can also calculate extinction rate per interval of time
freqRat(rangesDisc, calcExtinction = TRUE)

# est. ext rate = ~0.44 per interval
# 5 time-unit intervals, so ~0.44 / 5 = ~0.08 per time-unit
# That's pretty close to the generating value of 0.01, used in sampleRanges

## Not run:
#################
# The following example code (which is not run by default) examines how
# the freqRat estimates vary with sample size, interval length
# and compares it to using make_durationFreqDisc

# how good is the freqRat at 20 sampled taxa on avg?
set.seed(444)
r <- runif(100)
int.length = 1
# estimate R from r, assuming stuff like p = q
R <- sapply(r, sRate2sProb, int.length = 1)
ntaxa <- freqRats <- numeric()
for(i in 1:length(r)){
# assuming budding model
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          r = r[i],
                          nruns = 1,
                          nSamp = c(15,25),
                          nExtant = 0,
                          plot = TRUE
                          )
ranges <- fossilRecord2fossilRanges(record)
timeList <- binTimeData(ranges,int.length = int.length)
ntaxa[i] <- nrow(timeList[[2]])
freqRats[i] <- freqRat(timeList)
}
plot(R,freqRats);abline(0,1)
```

```
# without the gigantic artifacts bigger than 1...
plot(R,freqRats,ylim = c(0,1));abline(0,1)
# very worrisome lookin'!

# how good is it at 100 sampled taxa on average?
set.seed(444)
r <- runif(100)
int.length = 1
R <- sapply(r,sRate2sProb,int.length = 1)
ntaxa <- freqRats <- numeric()
for(i in 1:length(r)){
    # assuming budding model
    record <- simFossilRecord(p = 0.1,
                              q = 0.1,
                              r = r[i],
                              nruns = 1,
                              nSamp = c(80,150),
                              nExtant = 0,
                              plot = TRUE)
ranges <- fossilRecord2fossilRanges(record)
timeList <- binTimeData(ranges,
     int.length = int.length)
ntaxa[i] <- nrow(timeList[[2]])
freqRats[i] <- freqRat(timeList)
}

plot(R, freqRats,
     ylim = c(0,1)
     )
abline(0,1)

#not so hot, eh?

#################
#LETS CHANGE THE TIME BIN LENGTH!

# how good is it at 100 sampled taxa on average, with longer time bins?
set.seed(444)
r <- runif(100)
int.length <- 10
R <- sapply(r, sRate2sProb, int.length = int.length)
ntaxa <- freqRats <- numeric()
for(i in 1:length(r)){
   # assuming budding model
   record <- simFossilRecord(p = 0.1,
                             q = 0.1,
                             r = r[i],
                             nruns = 1,
                             nSamp = c(80,150),
                             nExtant = 0,
                             plot = TRUE)
   ranges <- fossilRecord2fossilRanges(record)
   timeList <- binTimeData(ranges, int.length = int.length)
```

```
   ntaxa[i] <- nrow(timeList[[2]])
   freqRats[i] <- freqRat(timeList)
   }

plot(R, freqRats, ylim = c(0,1))
abline(0,1)
# things get more accurate as interval length increases... odd, eh?

# how good is it at 20 sampled taxa on average, with longer time bins?
set.seed(444)
r <- runif(100)
int.length <- 10
R <- sapply(r, sRate2sProb, int.length = int.length)
ntaxa <- freqRats <- numeric()
for(i in 1:length(r)){
# assuming budding model
record <- simFossilRecord(p = 0.1,
                          q = 0.1,
                          r = r[i],
                          nruns = 1,
                          nSamp = c(15,25),
                          nExtant = 0,
                          plot = TRUE)
ranges <- fossilRecord2fossilRanges(record)
timeList <- binTimeData(ranges, int.length = int.length)
ntaxa[i] <- nrow(timeList[[2]])
freqRats[i] <- freqRat(timeList)
}
plot(R, freqRats, ylim = c(0,1))
abline(0,1)
# still not so hot at low sample sizes, even with longer bins

########################
# ML METHOD

# how good is the ML method at 20 taxa, 1 time-unit bins?
set.seed(444)
r <- runif(100)
int.length <- 1
R <- sapply(r,sRate2sProb,int.length = int.length)
ntaxa <- ML_sampProb <- numeric()
for(i in 1:length(r)){
   # assuming budding model
   record <- simFossilRecord(p = 0.1,
                             q = 0.1,
                             r = r[i],
                             nruns = 1,
                             nSamp = c(15,25),
                             nExtant = 0,
                             plot = TRUE
                             )
   ranges <- fossilRecord2fossilRanges(record)
   timeList <- binTimeData(ranges, int.length = int.length)
```

```
   ntaxa[i] <- nrow(timeList[[2]])
    likFun <- make_durationFreqDisc(timeList)
    ML_sampProb[i] <- optim(
         parInit(likFun), likFun,
      lower = parLower(likFun),
      upper = parUpper(likFun),
         method = "L-BFGS-B",
         control = list(maxit = 1000000)
         )[[1]][2]
   }

plot(R, ML_sampProb)
abline(0,1)

# Not so great due to likelihood surface ridges
 # but it returns values between 0-1

# how good is the ML method at 100 taxa, 1 time-unit bins?
set.seed(444)
r <- runif(100)
int.length <- 1
R <- sapply(r, sRate2sProb,
    int.length = int.length)
ntaxa <- ML_sampProb <- numeric()
for(i in 1:length(r)){
    # assuming budding model
   record <- simFossilRecord(p = 0.1,
                             q = 0.1,
                             r = r[i],
                             nruns = 1,
                             nSamp = c(80,150),
                             nExtant = 0,
                             plot = TRUE)
   ranges <- fossilRecord2fossilRanges(record)
   timeList <- binTimeData(ranges,int.length = int.length)
   ntaxa[i] <- nrow(timeList[[2]])
    likFun <- make_durationFreqDisc(timeList)
    ML_sampProb[i] <- optim(parInit(likFun),
                        likFun,
                    lower = parLower(likFun),
                    upper = parUpper(likFun),
                       method = "L-BFGS-B",
                       control = list(maxit = 1000000)
                       )[[1]][2]
}

plot(R,ML_sampProb)
abline(0,1)

# Oh, fairly nice, although still a biased uptick as R gets larger

## End(Not run)
```

getDataPBDB *Obtaining Data for Taxa or Occurrences From Paleobiology Database API*

Description

The Paleobiology Database API (link) is very easy to use, and generally any data one wishes to collect can be obtained in R through a variety of ways - the simplest being to wrap a data retrieval request to the API, specified for CSV output, with R function `read.csv`. The functions listed here, however, are some simple helper functions for doing tasks common to users of this package - downloading occurrence data, or taxonomic information, for particular clades, or for a list of specific taxa.

Usage

```
getCladeTaxaPBDB(
  taxon,
  showTaxa = c("class", "parent", "app", "img", "entname"),
  status = "accepted",
  urlOnly = FALSE,
  stopIfMissing = FALSE
)

getSpecificTaxaPBDB(
  taxa,
  showTaxa = c("class", "parent", "app", "img", "entname"),
  status = "accepted",
  urlOnly = FALSE,
  stopIfMissing = FALSE
)

getPBDBocc(
  taxa,
  showOccs = c("class", "classext", "subgenus", "ident", "entname")
)
```

Arguments

`taxon`	A single name of a of a higher taxon which you wish to catch all taxonomic 'children' (included members - i.e. subtaxa) of, from within the Paleobiology Database.
`showTaxa`	Which variables for taxonomic data should be requested from the Paleobiology Database? The default is to include classification (`"class"`), parent-child taxon information (`"parent"`), information on each taxon's first and last appearance (`"app"`), information on the PhyloPic silhouette images assigned to that taxon (`"img"`), and the names of those who entered and authorized the taxonomic data you have downloaded (`"entname"`). Multiple variable blocks can be

given as a single character string, with desired variable selections separated by a comma with no whitespace (ex. `"class,img,app"`) or as a vector of character strings (ex. `c("class","img","app")`), which will then formatted for use in the API call. Other options that you might want to include, such as information on ecospace or taphonomy, can be included: please refer to the full list at the documentation for the API.

`status` What taxonomic status should the pull taxa have? The default is `status = "accepted"`, which means only those taxa that are both valid taxa and *the accepted senior homonym*. Other typical statuses to consider are `"valid"`, which is all valid taxa: senior homonyms and valid subjective synonyms, and `"all"`, which will return all valid taxa and all otherwise repressed invalid taxa. For additional statuses that you can request, please see the documentation at the documentation for the API.

`urlOnly` If `FALSE` (the default), then the function behaves as expected, the API is called and a data table pulled from the Paleobiology Database is returned. If `urlOnly = TRUE`, the URL of the API call is returned instead as a character string.

`stopIfMissing` If some taxa within the requested set appear to be missing from the Paleobiology Database's taxonomy table, should the function halt with an error?

`taxa` A character vector listing taxa of interest that the user wishes to download information on from the Paleobiology Database. Multiple taxa can be listed as a single character string, with desired taxa separated by a comma with no whitespace (ex. `"Homo,Pongo,Gorilla"`) or as a vector of character strings (ex. `c("Homo","Pongo","Gorilla")`), which will then formatted for use in the API call.

`showOccs` Which variables for occurrence data should be requested from the Paleobiology Database? The default is to include classification (`"class"`), classification identifiers (`"classext"`), genus and subgenus identifiers (`"subgenus"`), and species-level identifiers (`"ident"`). Multiple variable blocks can be given as a single character string, with desired variable selections separated by a comma with no whitespace (ex. `"class,subgenus,ident"`) or as a vector of character strings (ex. `c("class","subgenus","ident")`), which will then formatted for use in the API call. For full list of other options that you might want to include, please refer to documentation for the API.

Details

In many cases, it might be easier to write your own query - these functions are only made to make getting data for some very specific applications in `paleotree` easier.

Value

These functions return a `data.frame` containing variables pulled for the requested taxon selection. This behavior can be modified by argument `urlOnly`.

Author(s)

David W. Bapst

References

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

See Also

See makePBDBtaxonTree, makePBDBtaxonTree, and plotPhyloPicTree for functions that use taxonomic data. Occurrence data is sorted by taxon via taxonSortPBDBocc, and further utilized occData2timeList and plotOccData.

Examples

```
#graptolites
graptData <- getCladeTaxaPBDB("Graptolithina")
dim(graptData)
sum(graptData$taxon_rank=="genus")

# so we can see that our call for graptolithina returned
# a large number of taxa, a large portion of which are
# individual genera
# (554 and 318 respectively, as of 03-18-19)

tetrapodList<-c("Archaeopteryx", "Columba", "Ectopistes",
"Corvus", "Velociraptor", "Baryonyx", "Bufo",
"Rhamphorhynchus", "Quetzalcoatlus", "Natator",
"Tyrannosaurus", "Triceratops", "Gavialis",
"Brachiosaurus", "Pteranodon", "Crocodylus",
"Alligator", "Giraffa", "Felis", "Ambystoma",
"Homo", "Dimetrodon", "Coleonyx", "Equus",
"Sphenodon", "Amblyrhynchus")

tetrapodData <-getSpecificTaxaPBDB(tetrapodList)
dim(tetrapodData)
sum(tetrapodData$taxon_rank=="genus")
# should be 26, with all 26 as genera

##########################################
# Now let's try getting occurrence data

# getting occurrence data for a genus, sorting it
# Dicellograptus
dicelloData <- getPBDBocc("Dicellograptus")
dicelloOcc2 <- taxonSortPBDBocc(dicelloData,
rank = "species", onlyFormal = FALSE)
names(dicelloOcc2)
```

graptDisparity *Morphlogical Character and Range Data for late Ordovician and Early Silurian Graptoloidea*

Description

This dataset contains a morphological character matrix (containing both a set of 45 discrete characters, and 4 continuous characters coded as minimum and maximum range values), along with biostratigraphic range data for 183 graptoloid species-level taxa from Bapst et al. (2012, PNAS). Also includes a pre-calculated distance matrix based on the character matrix, using the algorithm applied by Bapst et al (2012). Interval dates for biostratigraphic zones is taken from Sadler et al. 2011.

Format

This dataset is composed of three objects:

graptCharMatrix A `matrix` composed of mixed character data and a group code for 183 graptoloid taxa, with rows named with species names and columns named with character names.

graptRanges A `list` containing two matrices: the first matrix describes the first and last interval times for 20 graptolite biozones and the second matrix contains the first and last appearances of 183 graptolite species in those same biozones. (In other words, `graptRanges` has the `timeList` format called by some `paleotree` functions).

graptDistMat A 183x183 `matrix` of pair-wise distances (dissimilarities) for the 183 graptolite species, using the algorithm for discrete characters and min-max range values described in Bapst et al.

Details

The character matrix contains characters of two differing types with a (very) small but non-negligible amount of missing character data for some taxa. This required the use of an unconventional ad hoc distance metric for the published analysis, resulting in a (very slightly) non-Euclidean distance matrix. This breaks some assumptions of some statistical analyses or requires special corrections, such as with PCO.

Note that taxonomic data were collected only for species present within an interval defined by the base of the Uncinatus biozone (~448.57 Ma) to the end of the cyphus biozone (~439.37 Ma). Many taxa have first and last appearance dates listed in `graptRanges` which are outside of this interval (see examples).

Source

Source for stratigraphic ranges and character data:

Bapst, D. W., P. C. Bullock, M. J. Melchin, H. D. Sheets, and C. E. Mitchell. 2012. Graptoloid diversity and disparity became decoupled during the Ordovician mass extinction. *Proceedings of the National Academy of Sciences* 109(9):3428-3433.

Source for interval dates for graptolite zones:

Sadler, P. M., R. A. Cooper, and M. Melchin. 2009. High-resolution, early Paleozoic (Ordovician-Silurian) time scales. *Geological Society of America Bulletin* 121(5-6):887-906.

See Also

For more example graptolite datasets, see retiolitinae

This data was added mainly to provide an example dataset for nearestNeighborDist

Examples

```
#load data
data(graptDisparity)

#separate out two components of character matrix

#45 discrete characters
discChar <- graptCharMatrix[,1:45]

#min ranges for 4 continuous characters
cMinChar <- graptCharMatrix[,c(46,48,50,52)]
#max ranges for 4 continuous characters
cMaxChar <- graptCharMatrix[,c(47,49,51,53)]

#group (clade/paraclade) coding
groupID <- graptCharMatrix[,54]

#number of species
nspec <- nrow(graptCharMatrix)

#some plotting information from Bapst et al.'s plotting scripts
grpLabel <- c("Normalo.","Monogr.","Climaco.",
"Dicrano.","Lasiogr.","Diplogr.","Retiol.")
grpColor <- c("red","purple",colors()[257],colors()[614],
colors()[124],"blue",colors()[556])

##########

#plot diversity curve of taxa
taxicDivDisc(graptRanges)

#but the actual study interval for the data is much smaller
abline(v = 448.57,lwd = 3) #start of study interval
abline(v = 439.37,lwd = 3) #end of study interval

#plot diversity curve just for study interval
taxicDivDisc(graptRanges, timelims = c(448.57,439.37))

############

#distance matrix is given as graptDistMat
   #to calculate yourself, see code below in DoNotRun section
```

```
#now, is the diagonal zero? (it should be)
all(diag(graptDistMat) == 0)

#now, is the matrix symmetric? (it should be)
isSymmetric(graptDistMat)

#can apply cluster analysis
clustRes <- hclust(as.dist(graptDistMat))
plot(clustRes,labels = FALSE)

#use ape to plot with colors at the tips
dev.new(width = 15)  # for a prettier plot
plot.phylo(as.phylo(clustRes),show.tip.label = FALSE,
no.margin = TRUE,direction = "upwards")
tiplabels(pch = 16,col = grpColor[groupID+1])
legend("bottomright",legend = grpLabel,col = grpColor,pch = 16)
dev.set(2)

#can apply PCO (use lingoes correction to account for negative values
   #resulting from non-euclidean matrix
pco_res <- pcoa(graptDistMat,correction = "lingoes")

#relative corrected eigenvalues
rel_corr_eig <- pco_res$values$Rel_corr_eig
layout(1:2)
plot(rel_corr_eig)
#cumulative
plot(cumsum(rel_corr_eig))

#first few axes account for very little variance!!



#well let's look at those PCO axes anyway
layout(1)
pco_axes <- pco_res$vectors
plot(pco_axes[,1],pco_axes[,2],pch = 16,col = grpColor[groupID+1],
   xlab = paste("PCO Axis 1, Rel. Corr. Eigenvalue  = ",round(rel_corr_eig[1],3)),
   ylab = paste("PCO Axis 2, Rel. Corr. Eigenvalue  = ",round(rel_corr_eig[2],3)))
legend("bottomright",legend = grpLabel,col = grpColor,pch = 16,ncol = 2,cex = 0.8)


##########m###############

## Not run:

#calculate a distance matrix (very slow!)
#Bapst et al. calculated as # char diffs / total # of chars
   #but both calculated for only non-missing characters for both taxa
#non-identical discrete states = difference for discrete traits
#non-overlapping ranges for continuous characters = difference for cont traits
```

```
distMat <- matrix(,nspec,nspec)
rownames(distMat) <- colnames(distMat) <- rownames(graptCharMatrix)
for(i in 1:nspec){ for(j in 1:nspec){    #calculate for each pair of species
   #discrete characters
   di <- discChar[i,]     #discrete character vector for species i
   dj <- discChar[j,]     #discrete character vector for species j
   #now calculate pair-wise differences for non-missing characters
   discDiff <- (di != dj)[!is.na(di)&!is.na(dj)] #logical vector
   #
   #continuous characters: need another for() loop
   contDiff <- numeric()
   for(ct in 1:4){
       #if they do not overlap, a min must be greater than a max value
       contDiff[ct] <- cMinChar[i,ct]>cMaxChar[j,ct] | cMinChar[j,ct]>cMaxChar[i,ct]
       }
   #remove NAs
   contDiff <- contDiff[!is.na(contDiff)]
   #combine
   totalDiff <- c(discDiff,contDiff)
   #divide total difference
   distMat[i,j] <- sum(totalDiff)/length(totalDiff)
   }}

#but is it identical to the distance matrix already provided?
identical(distMat,graptDistMat)
#ehh, numerical rounding issues...

#A somewhat speeder alternative to calculate a distance matrix
distMat <- matrix(,nspec,nspec)
rownames(distMat) <- colnames(distMat) <- rownames(graptCharMatrix)
for(i in 1:(nspec-1)){ for(j in (i+1):nspec){    #calculate for each pair of species
   #now calculate pair-wise differences for non-missing characters
   discDiff <- (discChar[i,] != discChar[j,])[
      !is.na(discChar[i,])&!is.na(discChar[j,])] #logical vector
   #continuous characters: if they do not overlap, a min must be greater than a max value
      contDiff <- sapply(1:4,function(ct)
            cMinChar[i,ct]>cMaxChar[j,ct] | cMinChar[j,ct]>cMaxChar[i,ct])
   #remove NAs, combine, divide total difference
   distMat[i,j] <- distMat[j,i] <- sum(c(discDiff,contDiff[!is.na(contDiff)]))/length(
        c(discDiff,contDiff[!is.na(contDiff)]))
   }}
diag(distMat) <- 0

#but is it identical to the distance matrix already provided?
identical(distMat,graptDistMat)
#ehh, MORE numerical rounding issues...


## End(Not run)
```

graptPBDB *Example Occurrence and Taxonomic Datasets of the Graptolithina from the Paleobiology Database*

Description

Example datasets consisting of (a) occurrence data and (b) taxonomic data downloaded from the Paleobiology Database API for the Graptolithina. In order to make sure to catch anything that might be considered a graptolite, the actual taxon searched for was the Pterobranchia, the larger clade that includes graptolites within it (Mitchell et al., 2013).

Format

The example occurrence dataset (`graptOccPBDB`) is a `data.frame` consisting of 5900 occurrences (rows) and 35 variables (columns). The example taxonomy dataset (`graptTaxaPBDB`) is a `data.frame` consisting of 364 formal taxa (rows) and 53 variables (columns). Variables are coded in the 'pbdb' vocabulary of the PBDB API v1.2. Two phylogeny-like objects, an undated taxon-tree, and a dated version of the former, are provided as `graptTree` and `graptTimeTree` respectively.

Details

This example PBDB data is included here for testing functions involving occurrence data and taxonomy in `paleotree`.

Source

See examples for the full R code used to obtain the data from the API. You can find the Paleobiology Database at http://paleobiodb.org

The occurrence data was entered by many, including (six most prominent enterers, in order of relative portion): P. Novack-Gottshall, M. Krause, M. Foote, A. Hendy, T. Hanson, and M. Sommers. This entered data was authorized mainly by A. Miller, W. Kiessling, M. Foote, A. Hendy, S. Holland, J. Sepkoski (as well as others).

References

Mitchell, C. E., M. J. Melchin, C. B. Cameron, and J. Maletz. 2013. Phylogenetic analysis reveals that Rhabdopleura is an extant graptolite. *Lethaia* 46(1):34-56.

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

See Also

`taxonSortPBDBocc`, `occData2timeList`, `makePBDBtaxonTree`, `plotOccData`

Examples

```
# let's look for pterobranch genera
    # pterobranchs are the larger group containing graptolites

taxon <- "Pterobranchia"
selectRank <- "genus"

## Not run:
library(paleotree)

# get taxon data
    # default variables
graptTaxaPBDB<-getCladeTaxaPBDB(taxon)

# get the taxon tree
graptTree <- makePBDBtaxonTree(graptTaxaPBDB,
    rankTaxon = selectRank
    )

# date the tree using the ranges
    # provided directly by the PBDB
graptTimeTree <- dateTaxonTreePBDB(graptTree)

library(strap)
dev.new(height=6, width=10)
geoscalePhylo(graptTimeTree,
    ages=graptTimeTree$ranges.used
    )
nodelabels(graptTimeTree$node.label,
    cex=0.7,
    adj=c(0.3,0)
    )

# slice tree at the Mississippian-Pennslyvannian boundary so
    # the *two* extant genera don't obfuscate the tree
graptTimeTreePrePenn <- timeSliceTree(
    ttree = graptTimeTree,
    sliceTime = 323.2
    )
slicedRanges <- graptTimeTree$ranges.used
slicedRanges [slicedRanges < 323.2] <- 323.2

# plot it!
dev.new(height=6, width=10)
geoscalePhylo(graptTimeTreePrePenn,
    ages = slicedRanges
    )
nodelabels(graptTimeTreePrePenn$node.label,
    cex=0.7,
    adj=c(0.3,0)
    )
```

```
# we could also date the tree using the occurrence data
    # default variables
graptOccPBDB <- getPBDBocc(taxon)

# some PBDB people have names that aren't in ASCII
    # but CRAN hates non-ASCII character, sooo...
    # convert using gtools::ASCIIfy
levels(graptOccPBDB$enterer) <- gtools::ASCIIfy(
     levels(graptOccPBDB$enterer))
levels(graptOccPBDB$authorizer) <- gtools::ASCIIfy(
     levels(graptOccPBDB$authorizer))
levels(graptOccPBDB$modifier) <- gtools::ASCIIfy(
     levels(graptOccPBDB$modifier))

graptOccSort <- taxonSortPBDBocc(graptOccPBDB,
    rank = selectRank,
    onlyFormal = FALSE,
    cleanUncertain = FALSE)

graptTimeList <- occData2timeList(occList = graptOccSort)

graptTimeTreeFromOcc <- bin_timePaleoPhy(
    graptTree,
    timeList = graptTimeList,
    nonstoch.bin = TRUE,
    type = "mbl",
    vartime = 3)

plot(graptTimeTreeFromOcc, show.tip.label=FALSE)
axisPhylo()

# don't need to slice tree because extant-only taxa were dropped
dev.new(height=6, width=10)
geoscalePhylo(graptTimeTreeFromOcc,
    ages=graptTimeTreeFromOcc$ranges.used
   )
nodelabels(graptTimeTreeFromOcc$node.label,
    cex=0.7,
   adj=c(0.3,0)
   )

graphics.off()

save(graptOccPBDB,
     graptTaxaPBDB,
     graptTree,
     graptTimeTree,
     file = "graptPBDB.rdata")

## End(Not run)

# load archived example data
data(graptPBDB)
```

```
# let's visualize who entered the majority of the occurrence data
pie(sort(table(graptOccPBDB$enterer)))
# and now who authorized it
pie(sort(table(graptOccPBDB$authorizer)))

# I *sort of* apologize for using pie charts.

# Let's look at age resolution of these occurrences
hist(graptOccPBDB$max_ma - graptOccPBDB$min_ma,
    main = "Age Resolution of Occurrences",
    xlab = "Ma")

# use table to calculate distribution
    #of taxa among taxonomic ranks
table(graptTaxaPBDB$taxon_rank)

barplot(table(graptTaxaPBDB$taxon_rank))
```

`horizonSampRate` *Estimate Sampling Rate from Sampling Horizon Data (Solow and Smith, 1997)*

Description

This function implements the exact maximum likelihood estimator for the instantaneous sampling rate from Solow and Smith (1997, Paleobiology), which is based on the relationship between the number of collections for a set of taxa and their durations (known precisely in continuous time).

Usage

```
horizonSampRate(sampOcc = NULL, durations = NULL, nCollections = NULL)
```

Arguments

`sampOcc`	A list with the number of elements equal to the number of taxa, and each element of the list being a numerical vector with the length equal to the number of collections for each taxon, and each value equal to the precise date of that fossil's time of collection. These dates do not need to be ordered. If not supplied, the elements `durations` and `nCollections` must be supplied.
`durations`	A vector of precise durations in continuous time, with the length equal to the number of taxa. If not supplied, this is calculated from `SampOcc`, which must be supplied.
`nCollections`	A vector of integers representing the number of collections for each taxon in the input durations. If not supplied this is calculated from `SampOcc`, which must be supplied.

Details

Given a dataset of taxa with a vector N, representing the number of collections for each taxon, and a vector D, giving the precise duration for each taxon, we can use the following maximum likelihood estimator from Solow and Smith (1997) to obtain the instantaneous sampling rate:

$samplingRate = (sum(N-1)^2)/(sum(D) * sum(N))$

This method is exclusively for datasets with very precisely dated horizons, such as microfossils from deep sea cores with very precise age models. The first and last appearance must be known very precisely to provide an equally precise estimate of the duration. Most datasets are not precise enough for this method, due to chronostratigraphic uncertainty. However, note that the age of individual collections other than the first and last appearance dates do not need to be known: its only the number of collections that matters.

Value

Returns the instantaneous sampling (in per lineage*time-units) as a single numerical value. Note that this is the instantaneous sampling rate and not the probability of sampling a taxon per interval.

References

Solow, A. R., and W. Smith. 1997. On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* 23(3):271-277.

See Also

Duration frequency methods (Foote and Raup, 1996; Foote, 1997) use ranges alone to estimate sampling parameters, implemented in durationFreq.

Also see the conversion functions for sampling parameters at SamplingConv.

Examples

```
#can simulate this type of data with sampleRanges
    # just set ranges.only = FALSE
#let's try a simulation example:
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
sampledOccurrences <- sampleRanges(taxa,r = 0.5,ranges.only = FALSE)

# now try with horizonSampRate
horizonSampRate(sampOcc = sampledOccurrences)

# but we could also try with the *other* inputs
   # useful because some datasets we may only have durations
   # and number of sampling events for
filtered <- sampledOccurrences[!is.na(sampledOccurrences)]
dur <- sapply(filtered,max) - sapply(filtered,min)
nCol <- sapply(filtered,length)
# supply as durations and nCollections
horizonSampRate(durations = dur, nCollections = nCol)
```

inverseSurv *Inverse Survivorship Models in the Fossil Record*

Description

This function replicates the model-fitting procedure for forward and reverse survivorship curve data, described by Michael Foote in a series of papers (2001, 2003a, 2003b, 2005). These methods are discrete interval taxon ranges, as are used in many other functions in paleotree (see function arguments). This function can implement the continuous time, pulsed interval and mixed models described in Foote (2003a and 2005).

Usage

```
make_inverseSurv(
  timeList,
  groups = NULL,
  p_cont = TRUE,
  q_cont = TRUE,
  PA_n = "fixed",
  PB_1 = 0,
  Nb = 1,
  drop.extant = TRUE
)
```

Arguments

timeList	A two column matrix, with the first and last occurrences of taxa given in relative time intervals (i.e. ordered from first to last). If a list of `length = 2` is given for `timeData`, such as would be expected if the output of `binTimeData` was used as the input, the second element is used. See details. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as `NA`s in the second matrix) are automatically dropped from the `timeList` and from `groups` simultaneously. Living taxa observed in the modern day are expected to be listed as last observed in a special interval (`c(0,0)`), i.e. begins and ends at zero (modern) time. This interval is always automatically removed prior to the calculation intermediary data for fitting likelihood functions.
groups	Either NULL (the default) or matrix with the number of rows equal to the number of taxa and the number of columns equal to the number of 'systems' of categories for taxa. Taxonomic membership in different groups is indicated by numeric values. For example, a dataset could have a 'groups' matrix composed of a column representing thin and thick shelled taxa, coded `1` and `2` respectively, while the second column indicates whether taxa live in coastal, outer continental shelf, or deep marine settings, coded `1-3` respectively. Different combinations of groups will be treated as having independent sampling and extinction parameters in the default analysis, for example, thinly-shelled deep marine species will have separate parameters from thinly-shelled coastal species. Grouping systems could also represent temporal heterogeneity, for example, categorizing

Paleozoic versus Mesozoic taxa. If groups are NULL (the default), all taxa are assumed to be of the same group with the same parameters. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in timeData or timeList) are automatically dropped from groupings and the time dataset (either timeData or timeList) and from groups simultaneously.

p_cont — If TRUE (the default), then origination is assumed to be a continuous time process with an instantaneous rate. If FALSE, the origination is treated as a pulsed discrete-time process with a probability.

q_cont — If TRUE (the default), then extinction is assumed to be a continuous time process with an instantaneous rate. If FALSE, the extinction is treated as a pulsed discrete-time process with a probability.

PA_n — The probability of sampling a taxon after the last interval included in a survivorship study. Usually zero for extinct groups, although more logically has the value of 1 when there are still extant taxa (i.e., if the last interval is the Holocene and the group is still alive, the probability of sampling them later is probably 1...). Should be either be (a) a numeric value between 0 and 1, a NULL value, or can be simply be "fixed", the default option. This default PA_n = "fixed" option allows make_inverseSurv to decide the value based on whether there is a modern interval (i.e. an interval that is c(0,0)) or not: if there is one, then PA_n = 1, if not, then PA_n = 0. If PA_n = NULL, PA_n is treated as an additional free parameter in the output model.

PB_1 — The probability of sampling a taxon before the first interval included in a survivorship study. Should be a value of 0 to 1, or NULL. If NULL, PB_1 is treated as an additional free parameter in the output model.

Nb — The number of taxa that enter an interval (the b is for 'bottom'). This is an arbitrary constant used to scale other values in these calculations and can be safely set to 1.

drop.extant — Drops all extant taxa from a dataset before preceding.

Details

The design of this function to handle mixed continuous and discrete time models means that parameters can mean very different things, dependent on how a model is defined. Users should carefully evaluate their function arguments and the discussion of parameter values as described in the Value section.

Value

A function of class paleotreeFunc, which takes a vector equal to the number of parameters and returns the *negative* log likelihood (for use with optim and similar optimizing functions, which attempt to minimize support values). See the functions listed at modelMethods for manipulating and examining such functions and constrainParPaleo for constraining parameters.

The function output will take the largest number of parameters possible with respect to groupings and time-intervals, which means the number of parameters may number in the hundreds. Constraining the function for optimization is recommended except when datasets are very large.

Parameters in the output functions are named p, q and r, which are respectively the origination, extinction and sampling parameters. If the respective arguments p_cont and q_cont are TRUE, then

p and q will represent the instantaneous per-capita origination and extinction rates (in units of per lineage time-units). When one of these arguments is given as FALSE, the respective parameter (p or q) will represent per-lineage-interval rates. For p, this will be the per lineage-interval rate of a lineage producing another lineage (which can exceed 1 because diversity can more than double) and for q, this will be the per lineage-interval 'rate' of a lineage going extinct, which cannot be observed to exceed 1 (because the proportion of diversity that goes extinct *cannot* exceed 1). To obtain the per lineage-interval rates from a set of per lineage-time unit rates, simply multiply the per lineage-time-unit rate by the duration of an interval (or divide, to do the reverse; see Foote, 2003 and 2005). r is always the instantaneous per-capita sampling rate, in units per lineage-time units.

If PA_n or PB_1 were given as NULL in the arguments, two additional parameters will be added, named respectively PA_n and PB_1, and listed separately for every additional grouping. These are the probability of a taxon occurring before the first interval in the dataset (PB_1) and the probability of a taxon occurring after the last interval in a dataset (PA_n). Theses will be listed as PA_n.0 and PB_1.0 to indicate that they are not related to any particular time-interval included in the analysis, unlike the p, q, and r parameters (see below).

Groupings follow the parameter names, separated by periods; by default, the parameters will be placed in groups corresponding to the discrete intervals in the input timeList, such that make_inverseSurv will create a function with parameters p.1, q.1 and r.1 for interval 1; p.2, q.2 and r.2 for interval 2 and so on. Additional groupings given by the user are listed after this first set (e.g. 'p.1.2.2').

Calculating The Results of an Inverse Survivorship Model: Because of the complicated grouping and time interval scheme, combined with the probable preference of users to use constrained models rather that the full models, it may be difficult to infer what the rates for particular intervals and groups actually are in the final model, given the parameters that were found in the final optimization.

To account for this, the function output by inverseSurv also contains an alternative mode which takes input rates and returns the final values along with a rudimentary plotting function. This allows users to output per-interval and per-group parameter estimates. To select these feature, the argument altMode must be TRUE. This function will invisibly return the rate values for each group and interval as a list of matrices, with each matrix composed of the p, q and r rates for each interval, for a specific grouping.

This plotting is extremely primitive, and most users will probably find the invisibly returned rates to be of more interest. The function layout is used to play plots for different groupings in sequence, and this may lead to plots which are either hard to read or even cause errors (because of too many groupings, producing impossible plots). To repress this, the argument plotPar can be set to FALSE.

This capability means the function has more arguments that just the usual par argument that accepts the vector of parameters for running an optimization. The first of these additional arguments, altMode enables this alternative mode, instead of trying to estimate the negative log-likelihood from the given parameters. The other arguments augment the calculation and plotting of rates.

To summarize, a function output by inverseSurv has the following arguments:

par A vector of parameters, the same length as the number of parameters needed. For plotting, can be obtained with optimization

altMode If FALSE (the default) the function will work like ordinary model-fitting functions, returning a negative log-likelihood value for the input parameter values in par. If TRUE, however, the input parameters will instead be translated into the by-interval, by-group rates used for calculating the log-likelihoods, plotted (if plotPar is TRUE) and these final interval-specific rates will be returned invisibly as described above.

plotPar If TRUE (the default) the calculated rates will be plotted, with each grouping given a separate plot. This can be repressed by setting plotPar to FALSE. As the only conceivable purpose for setting plotPar to FALSE is to get the calculated rates, these will not be returned invisibly if plotPar is FALSE.

ratesPerInt If FALSE, the default option, the rates plotted and returned will be in units per lineage-time units, if those rates were being treated as rates for a continuous-time process (i.e. p_cont = TRUE and q_cont = TRUE for p and q, respectively, while r is always per lineage-time units). Otherwise, the respective rate will be in units per lineage-interval. If ratesPerInt is TRUE instead, then *all* rates, even rates modeled as continuous-time process, will be returned as per lineage-interval rates, even the sampling rate r.

logRates If FALSE (the default) rates are plotted on a linear scale. If TRUE, rates are plotted on a vertical log axis.

jitter If TRUE (default) the sampling rate and extinction rate will be plotted slightly ahead of the origination rate on the time axis, so the three rates can be easily differentiated. If FALSE, this is repressed.

legendPoisition The position of a legend indicating which line is which of the three rates on the resulting plot. This is given as the possible positions for argument x of the function legend, and by default is "topleft", which will be generally useful if origination and extinction rates are initially low. If legendPosition = NA, then a legend will not be plotted.

Note

This function is an entirely new rewrite of the methodology derived and presented by Foote in his studies. Thus, whether it would give identical results cannot be assumed nor is it easy to test given differences in the way data is handled between our coded functions. Furthermore, there may be differences in the math due to mistakes in the derivations caught while this function was programmed. I have tested the function by applying it to the same Sepkoski genus-level dataset that Foote used in his 2003 and 2005 papers. Users can feel free to contact me for detailed figures from this analysis. Overall, it seems my function captured the overall pattern of origination and sampling rates, at least under a model where both origination and extinction are modeled as continuous-time processes. Extinction showed considerably more variability relative to the published figures in Foote (2005). Additional analyses are being run to identify the sources of this discrepancy, and the function is being released here in paleotree on a trial basis, so that it can be more easily loaded onto remote servers. Users should be thus forewarned of this essentially 'beta' status of this function.

Author(s)

David W. Bapst, with some advice from Michael Foote.

References

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Foote, M. 2003a. Origination and Extinction through the Phanerozoic: A New Approach. *The Journal of Geology* 111(2):125-148.

Foote, M. 2003b. Erratum: Origination and Extinction through the Phanerozoic: a New Approach. *The Journal of Geology* 111(6):752-753.

Foote, M. 2005. Pulsed origination and extinction in the marine realm. *Paleobiology* 31(1):6-20.

See Also

This function extensively relies on footeValues.

A similar format for likelihood models can be seen in durationFreq.

Also see freqRat, sRate2sProb, qsRate2Comp sProb2sRate and qsProb2Comp.

For translating between sampling probabilities and sampling rates, see SamplingConv.

Examples

```
# let's simulate some taxon ranges from an imperfectly sampled fossil record
set.seed(444)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
   nTotalTaxa = c(30,40),
   nExtant = 0
   )
taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa, r = 0.5)

#bin the ranges into discrete time intervals
rangesDisc <- binTimeData(rangesCont, int.length = 5)

#apply make_inverseSurv
likFun <- make_inverseSurv(rangesDisc)

#use constrainParPaleo to make the model time-homogeneous
   # match.all ~ match.all will match parameters
   # so only 2 parameters: p (= q) and r

constrFun <- constrainParPaleo(likFun,
   match.all~match.all)

results <- optim(parInit(constrFun),
                 constrFun,
                 lower = parLower(constrFun),
                 upper = parUpper(constrFun),
                 method = "L-BFGS-B",
                 control = list(maxit = 1000000)
                 )
results

#plot the results
constrFun(results$par, altMode = TRUE)


## Not run:
#unconstrained function with ALL of the 225 possible parameters!!!
    # this will take forever to converge
```

```
optim(parInit(likFun),
      likFun,
      lower = parLower(likFun),
      upper = parUpper(likFun),
      method = "L-BFGS-B",
      control = list(maxit = 1000000)
      )

## End(Not run)
```

---

kanto *Example Species Abundances Tables*

Description

A totally fictional example of species abundance data, for testing functions that require a site-by-taxon table of community ecology data.

Format

A table of type integer, representing terrestrial fauna and flora abundance counts.

Details

A classic dataset of ecological data collected by Satoshi and Okido, consisting of individual counts for 54 terrestrial faunal and floral species, from 23 sites across the mainland Kanto region.

Different ontogenetic stages were compounded and recorded by the common name for the first ontogenetic stage, with some inconsistency for species whose earliest stage have only been recently recognized. When separate names are commonly applied to sexual dimorphic forms, these were also combined and a single common name was used.

Note: This data is a totally made-up, satirical homage to a well-known video game series (thus constituting fair-use).

Source

Pokemon And All Respective Names are Trademark and Copyright of Nintendo 1996-2015.

See Also

twoWayEcologyCluster, communityEcology

Examples

```
data(kanto)

#visualize site abundances as barplots
barplotAbund <- function(x){
```

```
x <- x[,colSums(x)>0]
layout(1:(nrow(x)+1))
xpar <- par(mar = c(0,7,2,0))
for(i in 1:(nrow(x)-1)){
barplot(x[i,],ylab = rownames(x)[i],
names.arg = "")
}
barplot(x[nrow(x),],
ylab = rownames(x)[nrow(x)],las = 3)
par(xpar)
layout(1)
mtext("Abundances",side = 2,line = 3,adj = 0.8)
}

#first five sites
kanto5 <- kanto[1:5,]
barplotAbund(kanto5)

#get pairwise Spearman rho coefficients
rhoCoeff <- pairwiseSpearmanRho(kanto,dropAbsent = "bothAbsent")

#what are the nearest-neighbor rhos (largest rho correlations)?
diag(rhoCoeff) <- NA
rhoNearest <- apply(rhoCoeff,1,max,na.rm = TRUE)
rhoNearest

# We can see the power plant sample is extremely different from the rest

# measure evenness: Hurlbert's PIE

kantoPIE <- HurlbertPIE(kanto)

# compare to dominance (relative abundance of most abundant taxon)

dominance <- apply(kanto,1,function(x) max(x)/sum(x) )

plot(kantoPIE,dominance)

# relatively strong relationship!


## Not run:
########################################
################################################
############################################################
# Some Cool Ecology Stuff With Other Packages

# basically all the analyses & visualizations
#for ecology in R that I think are awesome


########################################
############################
```

```
#Ordination (PCO, DCA)

#get bray-curtis distances
library(vegan)
bcDist <- vegdist(kanto,method = "bray")

# do a PCO on the bray-curtis distances
pcoRes <- pcoa(bcDist,correction = "lingoes")
scores <- pcoRes$vectors
# plot the PCO
plot(scores,type = "n")
text(labels = rownames(kanto),scores[,1],scores[,2],cex = 0.5)

# the way the power plant and the pokemon tower converge
# is very suspicious: may be distortion due to a long gradient

# do a DCA instead with vegan's decorana
dcaRes <- decorana(kanto)
# plot using native vegan functions
   #will show species scores in red
plot(dcaRes,cex = 0.5)
#kind of messy

#show just the sites scores
plot(dcaRes,cex = 0.5,display = "sites")

#show just the species scores
plot(dcaRes,cex = 0.5,display = "species")

#well, that's pretty cool

#######################
#get the nearest neighbor for each site
    # based on pair-wise rho coefficients
rhoNeighbor <- apply(rhoCoeff,1,function(x)
    rownames(kanto)[tail(order(x,na.last = NA),1)])

#let's plot the nearest neighbor connections with igraph
NNtable <- cbind(rownames(kanto),rhoNeighbor)

# now plot with igraph
library(igraph)
NNlist <- graph.data.frame(NNtable)
plot(NNlist)

#arrows point at the nearest neighbor of each sample
    # based on maximum Spearman rho correlation

########################################
#########################################################
# Two Way Cluster With Heatmap

# This example based on code provided by Max Christie
```

```
# load pheatmap library for this example
library(pheatmap)

# get distance matrices for sites and taxa
# based on bray-curtis dist
# standardized to total abundance

# standardize site matrix to relative abundance
siteStand <- decostand(kanto, method = "total")
# site distance matrix (Bray-Curtis)
siteDist <- vegdist(siteStand, "bray", diag = TRUE)

# standardize taxa matrix to relative abundance
taxaStand <- decostand(t(kanto), method = "total")
# taxa distance matrix (Bray-Curtis)
taxaDist <- vegdist(taxaStand, "bray", diag = TRUE)

### Need to set graphic parameters for table

# Check out range of values for relative abundance
# hist(myStand) # none get very high...

# number of breaks: number of colors for heatmap
nBreaks <- 15

# set underValue
# anything below this counts as not appearing
# at that site for visualization purposes
underValue <-  min(siteStand[siteStand>0])-min(siteStand[siteStand>0])/10
# set overValue (max relative abundance)
overValue <- max(siteStand)
# you can set your breaks to any sequence you want
# and they don't have to be the same length.
# You can do this manually too.
# here we added a 0 to 'underValue' bin to
# the heatmap, making this bin essentially 0.
colorBreaks <- c(0,seq(underValue,max(siteStand),
by = max(siteStand)/(nBreaks-1)))
# here we used the function rainbow to create a vector of colors.
# You can set these colors yourself too.
# It is important that this vector is one element
# less than the myBreaks vector
rainColors <- rainbow(nBreaks)
# now we can add "white" onto the vector,
# this will be the first color bin,
# which we're going to set to be (essentially) 0.
rainColors <- c("white", rainColors)
# If you don't add white, taxa at 0 abundance get colored in

### Plot the 2-Way Cluster

# heatmap, with user-set colors
```

```
# feed the function a distance matrix we wanted to use.
#siteDist and taxaDist made above by vegdist (bray-curtis)
# scale is the relative abundance, let's label it as such

dev.new(width = 10)

#for some reason, mtext() doesn't recognize pheatmap as plot.new
plot.new(width = 7)

pheatmap(
   siteStand,
  clustering_method = "ward.D",
  clustering_distance_rows = siteDist,
  clustering_distance_cols = taxaDist,
   color = rainColors,
   breaks = colorBreaks
   )
mtext("Relative Abundance",
   side = 4, line = -1.4, adj = 0.95)

# pretty cool looking!

########################
# even better:
   # twoWayEcologyCluster in paleotree

dev.new(width=10)

twoWayEcologyCluster(
    xDist = siteDist,
    yDist = taxaDist,
    propAbund = siteStandKanto,
    cex.axisLabels = 0.8
    )

#########################################
#########################################################
## Testing for differences between groups of sites

#is there a difference between routes and non-routes
groups <- rep(0, nrow(kanto))
groups[grep(rownames(kanto), pattern = "Route")] <- 1

#anosim (in vegan)
#are distances within groups smaller than distances between?
library(vegan)
anosim(dat = kanto, grouping = groups)

# we could also use PERMANOVA instead
    # this is generally considered more robust than ANOSIM
      # note that group needs to be factor for PERMANOVA
groupsAsFactor <- factor(groups)
adonis(kanto ~ groupsAsFactor)
```

```
# both analyses are very significant

#####################################################################
# SIMPER analysis (SIMalarity PERcentages) in Vegan
# which taxa contribute most to the difference between groups?
     # this might be 'index' taxa for different communities
# beware: it might also be the taxa that vary most within groups

simperResult <- simper(comm = kanto, group = groupsAsFactor)
simperResult

# these are the species that account for at least 70% of
# differences between groups, based on Bray-Curtis distances

# can see % contribtion for all species with summary()
    # as well as more detail in general...
summary(simperResult)

# other analyses to look into:
   # SimProf to test clusters from a cluster analysis...

#########################################################
# alternative for differentiating groups:
   # using multivariate GLMs in mvabund

library(mvabund)

ft <- manyglm(formula = kanto ~ groupsAsFactor)
anova(ft)

# also highly significant!
# note that this method though uses absolute abundances
# it will not accepted
    # which are usually impossible to get


## End(Not run)
```

`macroperforateForam` *Ancestor-Descendant Relationships for Macroperforate Foraminifera, from Aze et al. (2011)*

Description

An example dataset of ancestor-descendant relationships and first and last appearance dates for a set of macroperforate Foraminifera, taken from the supplemental materials of Aze et al. (2011). This dataset is included here primarily for testing functions `parentChild2taxonTree` and `taxa2phylo`.

Format

The `foramAM` and `foramAL` tables include budding taxon units for morphospecies and lineages respective, with four columns: taxon name, ancestral taxon's name, first appearance date and last appearance date (note that column headings vary). The `foramAMb` and `foramALb` tables are composed of data for the same taxon units as the previous branching events are split so that the relationships are fully 'bifurcating', rather than 'budding'. As this obscures taxonomic identity, taxon identification labels are included in an additional, fifth column in these tables. See the examples section for more details.

Details

This example dataset is composed of four tables, each containing information on the ancestor-descendant relationships and first and last appearances of species of macroperforate foraminifera species from the fossil record. Each of the four tables are for the same set of taxa, but divide and concatenate the included foram species in four different ways, relating to the use of morpospecies versus combined anagenetic lineages (see Ezard et al., 2012), and whether taxa are retained as units related by budding-cladogenesis or the splitting of taxa at branching points to create a fully 'bifurcating' set of relationships, independent of ancestral morphotaxon persistence through branching events. See the examples section for more details.

Source

This dataset is obtained from the supplementary materials of, specifically 'Appendix S5':

Aze, T., T. H. G. Ezard, A. Purvis, H. K. Coxall, D. R. M. Stewart, B. S. Wade, and P. N. Pearson. 2011. A phylogeny of Cenozoic macroperforate planktonic foraminifera from fossil data. *Biological Reviews* 86(4):900-927.

References

This dataset has been used or referenced in a number of works, including:

Aze, T., T. H. G. Ezard, A. Purvis, H. K. Coxall, D. R. M. Stewart, B. S. Wade, and P. N. Pearson. 2013. Identifying anagenesis and cladogenesis in the fossil record. *Proceedings of the National Academy of Sciences* 110(32):E2946-E2946.

Ezard, T. H. G., T. Aze, P. N. Pearson, and A. Purvis. 2011. Interplay Between Changing Climate and Species' Ecology Drives Macroevolutionary Dynamics. *Science* 332(6027):349-351.

Ezard, T. H. G., P. N. Pearson, T. Aze, and A. Purvis. 2012. The meaning of birth and death (in macroevolutionary birth-death models). *Biology Letters* 8(1):139-142.

Ezard, T. H. G., G. H. Thomas, and A. Purvis. 2013. Inclusion of a near-complete fossil record reveals speciation-related molecular evolution. *Methods in Ecology and Evolution* 4(8):745-753.

Strotz, L. C., and A. P. Allen. 2013. Assessing the role of cladogenesis in macroevolution by integrating fossil and molecular evidence. *Proceedings of the National Academy of Sciences* 110(8):2904-2909.

Strotz, L. C., and A. P. Allen. 2013. Reply to Aze et al.: Distinguishing speciation modes based on multiple lines of evidence. *Proceedings of the National Academy of Sciences* 110(32):E2947-E2947.

Examples

```
# Following Text Reproduced from Aze et al. 2011's Supplemental Material
# Appendix S5
#
# 'Data required to produce all of the phylogenies included in the manuscript
# using paleoPhylo (Ezard & Purvis, 2009) a free software package to draw
# paleobiological phylogenies in R.'
#
# 'The four tabs hold different versions of our phylogeny:
#  aMb: fully bifurcating morphospecies phylogeny
#  aM: budding/bifurcating morphospecies phylogeny
#  aLb: fully bifurcating lineage phylogeny
#  aL: budding/bifurcating lineage phylogeny
#
# 'Start Date gives the first occurence of the species according
# to the particular phylogeny; End Date gives the last occurence
# according to the particular phylogeny.'

## Not run:

# load the data
    # given in supplemental as XLS sheets
    # converted to separate tab-deliminated text files

# aM: budding/bifurcating morphospecies phylogeny
foramAM <- read.table(file.choose(),stringsAsFactors = FALSE,header = TRUE)
# aL: budding/bifurcating lineage phylogeny
foramAL <- read.table(file.choose(),stringsAsFactors = FALSE,header = TRUE)
# aMb: fully bifurcating morphospecies phylogeny
foramAMb <- read.table(file.choose(),stringsAsFactors = FALSE,header = TRUE)
# aLb: fully bifurcating lineage phylogeny
foramALb <- read.table(file.choose(),stringsAsFactors = FALSE,header = TRUE)

save.image("macroperforateForam.rdata")


## End(Not run)

# or instead, we'll just load the data directly
data(macroperforateForam)

#Two distinctions among the four datasets:
#(1): morphospecies vs morphospecies combined into sequences of anagenetic
# morpospecies referred to as 'lineages'. Thus far more morphospecies
# than lineages. The names of lineages are given as the sequence of
# their respective component morphospecies.
#(2): Datasets where taxon units (morphospecies or lineages) are broken up
# at 'budding' branching events (where the ancestral taxon persists)
# so that final dataset is 'fully bifurcating', presumably
# to make comparison easier to extant-taxon only datasets.
# (This isn't a limitation for paleotree, though!).
```

```
# This division of taxon units requires abstracting the taxon IDs,
# requiring another column for Species Name.

dim(foramAM)
dim(foramAL)
dim(foramAMb)
dim(foramALb)

#Need to convert these to same format as fossilRecord2fossilTaxa output.
#those 'taxa' tables has 6 columns:
#taxon.id ancestor.id orig.time ext.time still.alive looks.like

#for the purposes of this, we'll make taxon.id = looks.like
# (That's only for simulating cryptic speciation anyway)
#still.alive should be TRUE (1) if ext.time = 0

#a function to convert Aze et al's suppmat to paleotree-readable format

createTaxaData <- function(table){
#reorder table by first appearance time
table <- table[order(-as.numeric(table[,3])),]
ID <- 1:nrow(table)
anc <- sapply(table[,2],function(x)
if(!is.na(x)){
which(x == table[,1])
}else{ NA })
stillAlive <- as.numeric(table[,4] == 0)
ages <- cbind(as.numeric(table[,3]),as.numeric(table[,4]))
res <- cbind(ID,anc,ages,stillAlive,ID)
colnames(res) <- c('taxon.id','ancestor.id','orig.time',
'ext.time','still.alive','looks.like')
rownames(res) <- table[,1]
return(res)
}

taxaAM <- createTaxaData(foramAM)
taxaAMb <- createTaxaData(foramAMb)
taxaAL <- createTaxaData(foramAL)
taxaALb <- createTaxaData(foramALb)

##################################

#Checking Ancestor-Descendant Relationships for Irregularities

#For each of these, there should only be a single taxon
# without a parent listed (essentially, the root ancestor)

countParentsWithoutMatch <- function(table){
     parentMatch <- match(unique(table[,2]),table[,1])
     sum(is.na(parentMatch))
}

#test this on the provided ancestor-descendant relationships
```

```
countParentsWithoutMatch(foramAM)
countParentsWithoutMatch(foramAL)
countParentsWithoutMatch(foramAMb)
countParentsWithoutMatch(foramALb)

#and on the converted datasets
countParentsWithoutMatch(taxaAM)
countParentsWithoutMatch(taxaAL)
countParentsWithoutMatch(taxaAMb)
countParentsWithoutMatch(taxaALb)



#can construct the parentChild2taxonTree
#using the ancestor-descendant relationships

#can be very slow...

treeAM <- parentChild2taxonTree(foramAM[,2:1])
treeAL <- parentChild2taxonTree(foramAL[,2:1])
treeAMb <- parentChild2taxonTree(foramAMb[,2:1])
treeALb <- parentChild2taxonTree(foramALb[,2:1])

layout(matrix(1:4,2,2))
plot(treeAM,main = 'treeAM',show.tip.label = FALSE)
plot(treeAL,main = 'treeAL',show.tip.label = FALSE)
plot(treeAMb,main = 'treeAMb',show.tip.label = FALSE)
plot(treeALb,main = 'treeALb',show.tip.label = FALSE)

# FYI
# in case you were wondering
# you would *not* time-scale these Frankenstein monsters



##########################################

# Checking stratigraphic ranges

# do all first occurrence dates occur before last occurrence dates?
# we'll check the original datasets here

checkFoLo <- function(data){
diffDate <- data[,3]-data[,4] #subtract LO from FO
isGood <- all(diffDate >= 0) #is it good
return(isGood)
}

checkFoLo(foramAM)
checkFoLo(foramAL)
checkFoLo(foramAMb)
checkFoLo(foramALb)
```

```
#cool, but do all ancestors appear before their descendants?
# easier to check unified fossilRecord2fossilTaxa format here

checkAncOrder <- function(taxa){
#get ancestor's first occurrence
ancFO <- taxa[taxa[,2],3]
#get descendant's first occurrence
descFO <- taxa[,3]
diffDate <- ancFO-descFO #subtract descFO from ancFO
#remove NAs due to root taxon
diffDate <- diffDate[!is.na(diffDate)]
isGood <- all(diffDate >= 0) #is it all good
return(isGood)
}

checkAncOrder(taxaAM)
checkAncOrder(taxaAL)
checkAncOrder(taxaAMb)
checkAncOrder(taxaALb)

#now, are there gaps between the last occurrence of ancestors
# and the first occurrence of descendants?
# (shall we call these 'stratophenetic ghost branches'?!)
# These shouldn't be problematic, but do they occur in this data?
# After all, fossilRecord2fossilTaxa output tables are designed for
   # fully observed simulated fossil records with no gaps.

sumAncDescGap <- function(taxa){
#get ancestor's last occurrence
ancLO <- taxa[taxa[,2],4]
#get descendant's first occurrence
descFO <- taxa[,3]
diffDate <- ancLO-descFO #subtract descFO from ancFO
#remove NAs due to root taxon
diffDate <- diffDate[!is.na(diffDate)]
#should be negative or zero, positive values are gaps
gaps <- c(0,diffDate[diffDate>0])
sumGap <- sum(gaps)
return(sumGap)
}

#get the total gap between ancestor LO and child FO
sumAncDescGap(taxaAM)
sumAncDescGap(taxaAL)
sumAncDescGap(taxaAMb)
sumAncDescGap(taxaALb)

#It appears there is *no* gaps between ancestors and their descendants
#in the Aze et al. foram dataset... wow!

###############
```

```
# Creating time-scaled phylogenies from the Aze et al. data

# Aze et al. (2011) defines anagenesis such that taxa may overlap
# in time during a transitional period (see Ezard et al. 2012
# for discussion of this definition). Thus, we would expect that
# paleotree obtains very different trees for morphospecies versus
# lineages, but very similar phylogenies for datasets where budding
# taxa are retained or arbitrarily broken into bifurcating units.

# We can use the function taxa2phylo to directly create
# time-scaled phylogenies from the Aze et al. stratophenetic data

timetreeAM <- taxa2phylo(taxaAM)
timetreeAL <- taxa2phylo(taxaAL)
timetreeAMb <- taxa2phylo(taxaAMb)
timetreeALb <- taxa2phylo(taxaALb)

layout(matrix(1:4,2,2))
plot(timetreeAM,main = 'timetreeAM',show.tip.label = FALSE)
axisPhylo()
plot(timetreeAL,main = 'timetreeAL',show.tip.label = FALSE)
axisPhylo()
plot(timetreeAMb,main = 'timetreeAMb',show.tip.label = FALSE)
axisPhylo()
plot(timetreeALb,main = 'timetreeALb',show.tip.label = FALSE)
axisPhylo()

#visually compare the two pairs we expect to be close to identical

#morpospecies
layout(1:2)
plot(timetreeAM,main = 'timetreeAM',show.tip.label = FALSE)
axisPhylo()
plot(timetreeAMb,main = 'timetreeAMb',show.tip.label = FALSE)
axisPhylo()

#lineages
layout(1:2)
plot(timetreeAL,main = 'timetreeAL',show.tip.label = FALSE)
axisPhylo()
plot(timetreeALb,main = 'timetreeALb',show.tip.label = FALSE)
axisPhylo()

layout(1)

#compare the summary statistics of the trees
Ntip(timetreeAM)
Ntip(timetreeAMb)
Ntip(timetreeAL)
Ntip(timetreeALb)
# very different!
```

```
# after dropping anagenetic zero-length-terminal-edge ancestors
# we would expect morphospecies and lineage phylogenies to be very similar

#morphospecies
Ntip(dropZLB(timetreeAM))
Ntip(dropZLB(timetreeAMb))
#identical!

#lineages
Ntip(dropZLB(timetreeAL))
Ntip(dropZLB(timetreeALb))
# ah, very close, off by a single tip
# ...probably a very short ZLB outside tolerance

#we can create some diversity plots to compare

multiDiv(data = list(timetreeAM,timetreeAMb),
plotMultCurves = TRUE)

multiDiv(data = list(timetreeAL,timetreeALb),
plotMultCurves = TRUE)

# we can see that the morphospecies datasets are identical
# that's why we can only see one line
# some very slight disagreement between the lineage datasets
# around ~30-20 Ma

#can also compare morphospecies and lineages diversity curves

multiDiv(data = list(timetreeAM,timetreeAL),
plotMultCurves = TRUE)

#they are similar, but some peaks are missing from lineages
# particularly around ~20-10 Ma
```

`makePBDBtaxonTree` *Creating a Taxon-Tree from Taxonomic Data Downloaded from the Paleobiology Database*

Description

The function `makePBDBtaxonTree` creates phylogeny-like object of class `phylo` from the taxonomic information recorded in a taxonomy download from the PBDB for a given group. Two different algorithms are provided, the default being based on parent-child taxon relationships, the other based

on the nested Linnean hierarchy. The function plotTaxaTreePBDB is also provided as a minor helper function for optimally plotting the labeled topologies that are output by makePBDBtaxonTree.

Usage

```
makePBDBtaxonTree(
  taxaDataPBDB,
  rankTaxon,
  method = "parentChild",
  tipSet = NULL,
  cleanTree = TRUE,
  annotatedDuplicateNames = TRUE,
  APIversion = "1.2"
)

plotTaxaTreePBDB(taxaTree, edgeLength = 1)
```

Arguments

taxaDataPBDB	A table of taxonomic data collected from the Paleobiology Database, using the taxa list option with show = class. Should work with versions 1.1-1.2 of the API, with either the pbdb or com vocab. However, as accepted_name is not available in API v1.1, the resulting tree will have a taxon's *original* name and not any formally updated name.
rankTaxon	The selected taxon rank; must be one of 'species', 'genus', 'family', 'order', 'class' or 'phylum'.
method	Controls which algorithm is used for calculating the taxon-tree. The default option is method = "parentChild" which converts the listed binary parent-child taxon relationships in the Paleobiology Database- these parent-child relationships (if missing from the input dataset) are autofilled using API calls to the Paleobiology Database. Alternatively, users may use method = "Linnean", which converts the table of Linnean taxonomic assignments (family, order, etc as provided by show = class in PBDB API calls) into a taxon-tree. Two methods formerly both implemented under method = "parentChild" are also available as method = "parentChildOldMergeRoot" and method = "parentChildOldQueryPBDB" respectively. Both of these use similar algorithms as the current method = "parentChild" but differ in how they treat taxa with parents missing from the input taxonomic dataset. method = "parentChildOldQueryPBDB" behaves most similar to method = "parentChild" in that it queries the Paleobiology Database via the API , but repeatedly does so for information on parent taxa of the 'floating' parents, and continues within a while loop until only one such unassigned parent taxon remains. This latter option may talk a long time or never finish, depending on the linearity and taxonomic structures encountered in the PBDB taxonomic data; i.e. if someone a taxon was ultimately its own indirect child in some grand loop by mistake, then under this option makePBDBtaxonTree might never finish. In cases where taxonomy is bad due to weird and erroneous taxonomic assignments reported by the PBDB, this routine may search all the way back to a very ancient and deep taxon, such as the *Eukaryota* taxon. method =

"parentChildOldMergeRoot" will combine these disparate potential roots and link them to an artificially-constructed pseudo-root, which at least allows for visualization of the taxonomic structure in a limited dataset. This latter option will be fully offline, as it does not do any additional API calls of the Paleobiology Database, unlike other options.

tipSet — This argument only impacts analyses where method = "parentChild" is used. This tipSet argument controls which taxa are selected as tip taxa for the output tree. tipSet = "nonParents" selects all child taxa which are not listed as parents in parentChild. Alternatively, tipSet = "all" will add a tip to every internal node with the parent-taxon name encapsulated in parentheses. The default is NULL - if tipSet = NULL and method = "parentChild", then tipSet will be set so tipSet = "nonParents".

cleanTree — When TRUE (the default), the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If FALSE, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.

annotatedDuplicateNames — A logical determining whether duplicate taxon names, when found in the Paleobiology Database for taxa (presumably reflecting an issue with taxa being obsolete but with incomplete seniority data), should be annotated to include sequential numbers so to modify them, via functionbase's make.unique. This only applies to method = "parentChild", with the default option being annotatedDuplicateNames = TRUE. If more than 26 duplicates are found, an error is issued. If this argument is FALSE, an error is issued if duplicate taxon names are found.

APIversion — Version of the Paleobiology Database API used by makePBDBtaxonTree when method = "parentChild" or method = "parentChildOldQueryPBDB" is used. The current default is APIversion = "1.2", the most recent API version as of 12/11/2018.

taxaTree — A phylogeny of class phylo, presumably a taxon tree as output from makePBDBtaxonTree with higher-taxon names as node labels.

edgeLength — The edge length that the plotted tree should be plotted with (plotTaxaTreePBDB plots phylogenies as non-ultrametric, not as a cladogram with aligned tips).

Details

This function should not be taken too seriously. Many groups in the Paleobiology Database have out-of-date or very incomplete taxonomic information. This function is meant to help visualize what information is present, and by use of time-scaling functions, allow us to visualize the intersection of temporal and phylogenetic, mainly to look for incongruence due to either incorrect taxonomic placements, erroneous occurrence data or both.

Note however that, contrary to common opinion among some paleontologists, taxon-trees may be just as useful for macroevolutionary studies as reconstructed phylogenies (Soul and Friedman, 2015).

Value

A phylogeny of class `phylo`, where each tip is a taxon of the given `rankTaxon`. See additional details regarding branch lengths can be found in the sub-algorithms used to create the taxon-tree by this function: `parentChild2taxonTree` and `taxonTable2taxonTree`.

Depending on the `method` used, either the element `$parentChild` or `$taxonTable` is added to the list structure of the output phylogeny object, which was used as input for one of the two algorithms mentioned above.

Please note that when applied to output from the taxa option of the API version 1.1, the taxon names returned are the *original* taxon names as 'accepted_name' is not available in API v1.1, while under API v1.2, the returned taxon names should be the most up-to-date formal names for those taxa. Similar issues also effect the identification of parent taxa, as the accepted name of the parent ID number is only provided in version 1.2 of the API.

Author(s)

David W. Bapst

References

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

Soul, L. C., and M. Friedman. 2015. Taxonomy and Phylogeny Can Yield Comparable Results in Comparative Palaeontological Analyses. *Systematic Biology* (Link)

See Also

Two other functions in paleotree are used as sub-algorithms by `makePBDBtaxonTree` to create the taxon-tree within this function, and users should consult their manual pages for additional details:

`parentChild2taxonTree` and `taxonTable2taxonTree`

Closely related functions for

Other functions for manipulating PBDB data can be found at `taxonSortPBDBocc`, `occData2timeList`, and the example data at `graptPBDB`.

Examples

```
set.seed(1)


#get some example occurrence and taxonomic data
data(graptPBDB)

#get the taxon tree: Linnean method
graptTreeLinnean <- makePBDBtaxonTree(
    taxaDataPBDB = graptTaxaPBDB,
    rankTaxon = "genus",
    method = "Linnean")

#get the taxon tree: parentChild method
```

```
graptTreeParentChild <- makePBDBtaxonTree(
    taxaDataPBDB = graptTaxaPBDB,
    rankTaxon = "genus",
    method = "parentChild")

# let's plot these and compare them!
plotTaxaTreePBDB(graptTreeParentChild)

plotTaxaTreePBDB(graptTreeLinnean)

# pause 3 seconds so we don't spam the API
Sys.sleep(3)

#####################################################
# let's try some other groups

###################################
#conodonts
conoData <- getCladeTaxaPBDB("Conodonta")
conoTree <- makePBDBtaxonTree(
    taxaDataPBDB = conoData,
    rankTaxon = "genus",
    method = "parentChild")
# plot it!
plotTaxaTreePBDB(conoTree)

# pause 3 seconds so we don't spam the API
Sys.sleep(3)

############################
#asaphid trilobites
asaData <- getCladeTaxaPBDB("Asaphida")
asaTree <- makePBDBtaxonTree(
    taxaDataPBDB = asaData,
    rankTaxon = "genus",
    method = "parentChild")
# plot it!
plotTaxaTreePBDB(asaTree)

# pause 3 seconds so we don't spam the API
Sys.sleep(3)

##############################
#Ornithischia
ornithData <- getCladeTaxaPBDB("Ornithischia")
ornithTree <- makePBDBtaxonTree(
    taxaDataPBDB = ornithData,
    rankTaxon = "genus",
    method = "parentChild")
plotTaxaTreePBDB(ornithTree)

# pause 3 seconds so we don't spam the API
Sys.sleep(3)
```

```
#try Linnean!

#but first... need to drop repeated taxon first: Hylaeosaurus
    # actually this taxon seems to have been repaired
    # as of September 2019 !
# findHylaeo <- ornithData$taxon_name == "Hylaeosaurus"
# there's actually only one accepted ID number
# HylaeoIDnum <- unique(ornithData[findHylaeo,"taxon_no"])
# HylaeoIDnum
# so, take which one has occurrences listed
# dropThis <- which((ornithData$n_occs < 1) & findHylaeo)
# ornithDataCleaned <- ornithData[-dropThis,]

ornithTree <- makePBDBtaxonTree(
    ornithData,
    rankTaxon = "genus",
    method = "Linnean")
plotTaxaTreePBDB(ornithTree)

# pause 3 seconds so we don't spam the API
Sys.sleep(3)

#########################
# Rhynchonellida
rynchData <- getCladeTaxaPBDB("Rhynchonellida")
rynchTree <- makePBDBtaxonTree(
    taxaDataPBDB = rynchData,
    rankTaxon = "genus",
    method = "parentChild")
plotTaxaTreePBDB(rynchTree)

#some of these look pretty messy!
```

`minBranchLength` *Scales Edge Lengths of a Phylogeny to a Minimum Branch Length*

Description

Rescales a tree with edge lengths so that all edge lengths are at least some minimum branch length (sometimes abbreviated as "MBL" or "mbl"). Edge lengths are transformed so they are greater than or equal to the input minimum branch length, by subtracting edge length from more root-ward edges and added to later branches. This may or may not change the age of the root divergence, depending on the distribution of short branch lengths close to the root.

Usage

```
minBranchLength(tree, mbl, modifyRootAge = TRUE)
```

Arguments

tree	A phylogeny with edge lengths of class phylo.
mbl	The minimum branch length
modifyRootAge	If TRUE (the default), the input tree is checked for a root age given as $root.time and if present it is checked and fixed for any possible movement backwards due to short branches close to the root node.

Details

This function was formally an internal segment in timePaleoPhy, and now is called by timePaleoPhy instead, allowing users to apply minBranchLength to trees that already have edge lengths.

Value

A phylogeny with edge lengths of class phylo.

Author(s)

David W. Bapst

See Also

This function was originally an internal piece of timePaleoPhy, which implements the minimum branch length time-scaling method along with others, which may be what you're looking for (instead of this miscellaneous function).

Examples

```
#simulation with an example non-ultrametric tree

tree <- rtree(20)
# randomly replace edges with ZLBs
   # similar to multi2di output
tree <- degradeTree(tree,0.3,
   leave.zlb = TRUE)

tree2 <- minBranchLength(tree,0.1)

layout(1:2)

plot(tree)
axisPhylo()
plot(tree2)
axisPhylo()

layout(1)


#now let's try it with an ultrametric case
```

```
# get a random tree
tree <- rtree(30)
# randomly replace edges with ZLBs
   # similar to multi2di output
tree <- degradeTree(tree,0.5,leave.zlb = TRUE)
# now randomly resolve
tree <- di2multi(tree)
# give branch lengths so its ultrametric
tree <- compute.brlen(tree)

# and we have an ultrametric tree with polytomies, yay!
plot(tree)

# now randomly resolve
tree2 <- multi2di(tree)
# get new branch lengths as would with real data
tree2 <- minBranchLength(tree2,0.1)

layout(1:2)
plot(tree,show.tip.label = FALSE)
axisPhylo()
plot(tree2,show.tip.label = FALSE)
axisPhylo()

layout(1)

# check that root ages aren't being left unmodified
   # create a tree with lots of ZBLs at the root
x <- stree(10)
x$edge.length <- runif(Nedge(x))
x <- multi2di(x)
# give it a root age
x$root.time <- max(node.depth.edgelength(x))

z <- minBranchLength(tree = x, mbl = 1)
plot(z)
```

`minCharChange` *Estimating the Minimum Number of Character Transitions Using Maximum Parsimony*

Description

`minCharChange` is a function which takes a cladogram and a discrete trait and finds the solutions of inferred character states for ancestral nodes that minimizes the number of character state transitions (either gains or losses/reversals) for a given topology and a set of discrete character data. `minCharChange` relies on `ancPropStateMat`, which is a wrapper for `phangorn`'s function `ancestral.pars`.

Usage

```
minCharChange(
  trait,
  tree,
  randomMax = 10000,
  maxParsimony = TRUE,
  orderedChar = FALSE,
  type = "MPR",
  cost = NULL,
  printMinResult = TRUE,
  ambiguity = c(NA, "?"),
  dropAmbiguity = FALSE,
  polySymbol = "&",
  contrast = NULL
)

ancPropStateMat(
  trait,
  tree,
  orderedChar = FALSE,
  type = "MPR",
  cost = NULL,
  ambiguity = c(NA, "?"),
  dropAmbiguity = FALSE,
  polySymbol = "&",
  contrast = NULL,
  returnContrast = FALSE
)
```

Arguments

trait	A vector of trait values for a discrete character, preferably named with taxon names identical to the tip labels on the input tree.
tree	A cladogram of type phylo. Any branch lengths are ignored.
randomMax	The maximum number of cladograms examined when searching a large number of solutions consistent with the reconstructed ancestral states from ancestral.pars with the minimum number of character state transitions. If the number of potential solutions is less than randomMax, then solutions are exhaustively searched.
maxParsimony	If TRUE (the default), then only solutions with the smallest number of total transitions examined will be returned. Note that since solutions are stochastically 'guessed' at, and the number of possible solutions may not be exhaustively searched, there may have been solutions not examined with a lower number of transitions even if maxParsimony = TRUE. Regardless, one may want to do maxParsimony = FALSE if one is interested in whether there are solutions with a smaller number of gains or losses and thus wants to return all solutions.
orderedChar	If TRUE (not the default), then the character will be reconstructed with a cost (step) matrix of a linear, ordered character. This is not applicable if type =

"ACCTRAN", as cost matrices cannot be used with ACCTRAN in ancestral.pars, and an error will be returned if orderedChar = TRUE but a cost matrix is given, as the only reason to use orderedChar is to produce a cost matrix automatically.

type — The parsimony algorithm applied by ancestral.pars, which can apply one of two: "MPR" (the default) is a relatively fast algorithm developed by Hanazawa et al. (1995) and Narushima and Hanazawa (1997), which relies on reconstructing the states at each internal node by re-rooting at that node. "ACCTRAN", the "accelerated transitions" algorithm (Swofford and Maddison, 1987), favors character reversal over independent gains when there is ambiguity. The "ACCTRAN" option in ancestral.pars avoids repeated rerooting of the tree to search for a smaller set of maximum-parsimony solutions that satisfy the "ACCTRAN" algorithm, but does so by assigning edge weights. As of phangorn v1.99-12, "MPR" is calculated using the Sankoff parsimony algorithm, which allows multifurcations (polytomies). This is *not* true for "ACCTRAN", which uses the Fitch algorithm, which does not allow for multifurcations. An error is returned if a tree with multifurcations is passed and a user tries type = "ACCTRAN".

cost — A matrix of the cost (i.e. number of steps) necessary to change between states of the input character trait. If NULL (the default), the character is assumed to be unordered with equal cost to change from any state to another. Cost matrices only impact the "MPR" algorithm; if a cost matrix is given but type = "ACCTRAN", an error is issued.

printMinResult — If TRUE (the default), a summary of the results is printed to the terminal. The information in this summary may be more detailed if the results of the analysis are simpler (i.e. fewer unique solutions).

ambiguity — A vector of values which indicate ambiguous (i.e. missing or unknown) character state codings in supplied trait data. Taxa coded ambiguously as treated as being equally likely to be any state coding. By default, NA values and "?" symbols are treated as ambiguous character codings, in agreement with behavior of functions in packages phangorn and Claddis. This argument is designed to mirror an hidden argument with an identical name in function phyDat in package phangorn.

dropAmbiguity — A logical. If TRUE (which is not the default), all taxa with ambiguous codings as defined by argument ambiguity will be dropped prior to ancestral nodes being inferred. This may result in too few taxa.

polySymbol — A single symbol which separates alternative states for polymorphic codings; the default symbol is "&", following the output by Claddis's ReadMorphNexus function, where polymorphic taxa are indicated by default with a string with state labels separated by an "&" symbol. For example, a taxon coded as polymorphic for states 1 or 2, would be indicated by the string "1&2". polySymbol is used to break up these strings and automatically construct a fitting contrast table for use with this data, including for ambiguous character state codings.

contrast — A matrix of type integer with cells of 0 and 1, where each row is labeled with a string value used for indicating character states in trait, and each column is labeled with the formal state label to be used for assign taxa to particular character states. A value of 1 indicates that the respective coding string for that row should be interpreted as reflecting the character state listed for that column. A

coding could reflect multiple states (such as might occur when taxa are polymorphic for some morphological character), so the sums of rows and columns can sum to more than 1. If `contrast` is not `NULL` (the default), the arguments will nullify This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package phangorn. This structure is based on `phangorn`'s use of `contrasts` table used for statistical evaluation of factors. See the `phangorn` vignette "Special features of phangorn" for more details on its implementation within `phangorn` including an example. See examples below for the construction of an example contrast matrix for character data with polymorphisms, coded as character data output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated with a string with state labels separated by an `"&"` symbol.

`returnContrast` If `TRUE`, the contrast table used by `ancestral.pars` will be output instead for user evaluation that polymorphic symbols and ambiguous states are being parsed correctly.

Details

The wrapper function `ancPropStateMat` simply automates the application of functions `ancestral.pars` and `phyDat` from `phangorn`, along with several additional checks and code to present the result as a matrix, rather than a specialized list.

Note that although the default `orderedChar` argument assumes that multistate characters are unordered, the results of character change will always be reported as gains and losses relative to the numbering of the states in the output `transitionSumChanges`, exactly as if they had been ordered. In the case where the character is actually ordered, this may be considered a conservative approach, as using a parsimony algorithm for unordered character states allows fewer gains or losses to be counted on branches where multiple gains and losses are reported. If the character is presumably unordered *and multistate*, however, then the gains and losses division is *arbitrary nonsense* and should be combined to to obtain the total number of character changes.

Value

By default, `ancPropStateMat` returns a matrix, with rows corresponding to the ID numbers of tips and nodes in `$edge`, and columns corresponding to character states, with the value representing the proportional weight of that node being that state under the algorithm used (known tip values are always 1). If argument `returnContrast` is `TRUE` then `ancPropStateMat` will instead return the final contrast table used by `phyDat` for interpreting character state strings.

`minCharChange` invisibly returns a list containing the following elements, several of which are printed by default to the console, as controlled by argument `printMinResult`:

`message` Describes the performance of `minCharChange` at searching for a minimum solution.

`sumTransitions` A vector recording the total number of necessary transitions (sum total of gains and losses/reversal) for each solution; effectively the parsimony cost of each solution.

`minTransitions` A symmetrical matrix with number of rows and columns equal to the number of character states, with values in each cell indicating the minimum number of transitions from one ancestral state (i.e. the rows) to a descendant state (i.e. the columns), taken across the set of kept solutions (dependent on which are kept as decided by argument `maxParsimony`). Generally guaranteed not to add up to the number of edges contained within the input tree,

and thus may not represent any realistic evolutionary scenario but does represent a conservative approach for asking 'what is the smallest possible number of transitions from 0 to 1' or 'smallest possible number of transitions from 1 to 0', independently of each other.

`solutionArray` A three-dimensional array, where for each solution, we have a matrix with edges for rows and two columns indicating the ancestral and child nodes of that edge, with values indicating the states inferred for those nodes in a particular solution.

`transitionArray` A labeled three-dimensional array where for each solution we have a symmetrical matrix with number of rows and columns equal to the number of character states, with values in each cell indicating the total number of transitions from one ancestral state (i.e. the rows) to a descendant state (i.e. the columns).

`transitionSumChanges` Which is a three column matrix with a row for every solution, with the values in the three columns measuring the number of edges (branches) inferred to respectively have gains, no change or losses (i.e. reversals), as calculated relative to the order of character states.

Author(s)

David W. Bapst

References

Hanazawa, M., H. Narushima, and N. Minaka. 1995. Generating most parsimonious reconstructions on a tree: A generalization of the Farris-Swofford-Maddison method. Discrete Applied Mathematics 56(2-3):245-265.

Narushima, H., and M. Hanazawa. 1997. A more efficient algorithm for MPR problems in phylogeny. Discrete Applied Mathematics 80(2-3):231-238.

Schliep, K. P. 2011. phangorn: phylogenetic analysis in R. *Bioinformatics* 27(4):592-593.

Swofford, D. L., and W. P. Maddison. 1987. Reconstructing ancestral character states under Wagner parsimony. *Mathematical Biosciences* 87(2):199-229.

See Also

The functions described here are effectively wrappers of phangorn's function `ancestral.pars`.

Examples

```
# let's write a quick & dirty ancestral trait plotting function

quickAncPlotter <- function(tree, ancData, cex){
    ancCol <- (1:ncol(ancData))+1
        plot(tree,
           show.tip.label = FALSE,
           no.margin = TRUE,
           direction = "upwards")
        tiplabels(pch = 16,
           pie = ancData[(1:Ntip(tree)),],
           cex = cex,
           piecol = ancCol,
```

```
            col = 0)
    nodelabels(pie = ancData[-(1:Ntip(tree)),],
        cex = cex,
        piecol = ancCol)
    }

# example with retiolitid graptolite data

data(retiolitinae)

#unordered, MPR
ancMPR <- ancPropStateMat(retioTree,
        trait = retioChar[,2],
        type = "MPR")
quickAncPlotter(retioTree,
   ancMPR, cex = 0.5)
text(x = 4,y = 5,
   "type = 'MPR'", cex = 1.5)

minCharChange(retioTree,
        trait = retioChar[,2],
        type = "MPR")

# with simulated data

set.seed(444)
tree <- rtree(50)
#simulate under a likelihood model
char <- rTraitDisc(tree,
        k = 3, rate = 0.7)
tree$edge.length <- NULL
tree <- ladderize(tree)

#unordered, MPR
ancMPR <- ancPropStateMat(tree,
        trait = char,
        type = "MPR")
#unordered, ACCTRAN
ancACCTRAN <- ancPropStateMat(tree,
        trait = char,
        type = "ACCTRAN")
#ordered, MPR
ancMPRord <- ancPropStateMat(tree,
        trait = char,
        orderedChar = TRUE,
        type = "MPR")

 #let's compare MPR versus ACCTRAN results
 layout(1:2)
 quickAncPlotter(tree,
        ancMPR, cex = 0.3)
 text(x = 8, y = 15,
        "type = 'MPR'", cex = 1.5)
```

```
 quickAncPlotter(tree,
        ancACCTRAN, cex = 0.3)
 text(x = 9, y = 15,
        "type = 'ACCTRAN'",cex = 1.5)

 # MPR has much more uncertainty in node estimates
 # but that doesn't mean ACCTRAN is preferable

#let's compare unordered versus ordered under MPR
layout(1:2)
quickAncPlotter(tree,
        ancMPR, cex = 0.3)
text(x = 8, y = 15,
        "unordered char\nMPR", cex = 1.5)
quickAncPlotter(tree,
        ancMPRord,cex = 0.3)
text(x = 9, y = 15,
        "ordered char\nMPR", cex = 1.5)
layout(1)


## Not run:
# what ancPropStateMat automates (with lots of checks):

require(phangorn)
char1 <- matrix(char,,1)
rownames(char1) <- names(char)
#translate into something for phangorn to read
char1 <- phyDat(char1,
        type = "USER",
        levels = sort(unique(char1))
        )
x <- ancestral.pars(tree,
        char1,type = "MPR")
y <- ancestral.pars(tree,
        char1,type = "ACCTRAN")

## End(Not run)

#estimating minimum number of transitions with MPR
minCharChange(tree,
        trait = char,
        type = "MPR")

 # and now with ACCTRAN
 minCharChange(tree,
        trait = char,
        type = "ACCTRAN")

#POLYMORPHISM IN CHARACTER DATA


# example trait data with a polymorphic taxon
```

```
     # separated with '&' symbol
# similar to polymorphic data output by ReadMorphNexus from package Claddis
charPoly <- as.character(
        c(1,2,NA,0,0,1,"1&2",
            2,0,NA,0,2,1,1,"1&2")
            )
#simulate a tree with 16 taxa
set.seed(444)
tree <- rtree(15)
tree$edge.length <- NULL
tree <- ladderize(tree)
names(charPoly) <- tree$tip.label
charPoly

# need a contrast matrix that takes this into account
    #can build row by row, by hand

#first, build contrast matrix for basic states
contrast012 <- rbind(c(1,0,0),
                     c(0,1,0),
                     c(0,0,1))
colnames(contrast012) <- rownames(contrast012) <- 0:2
contrast012

#add polymorphic state and NA ambiguity as new rows
contrastPoly <- c(0,1,1)
contrastNA <- c(1,1,1)
contrastNew <- rbind(contrast012,
        '1&2' = contrastPoly,
        contrastNA)
rownames(contrastNew)[5] <- NA

#let's look at contrast
contrastNew

# now try this contrast table we've assembled
    # default: unordered, MPR
ancPoly <- ancPropStateMat(tree,
        trait = charPoly,
        contrast = contrastNew)

# but...!
# we can also do it automatically,
    # by default, states with '&' are automatically treated
    # as polymorphic character codings by ancPropStateMat
ancPolyAuto <- ancPropStateMat(tree,
        trait = charPoly,
        polySymbol = "&")

# but does this match what the table we constructed?
ancPropStateMat(tree,
        trait = charPoly,
   polySymbol = "&",
```

```
        returnContrast = TRUE)

# compare to contrastNew above!
# only difference should be the default ambiguous
# character '?' is added to the table

#compare reconstructions
layout(1:2)
quickAncPlotter(tree,
        ancPoly, cex = 0.5)
text(x = 3.5, y = 1.2,
        "manually-constructed\ncontrast", cex = 1.3)
quickAncPlotter(tree,
        ancPolyAuto, cex = 0.5)
text(x = 3.5, y = 1.2,
        "auto-constructed\ncontrast", cex = 1.3)
layout(1)

# look pretty similar!

# i.e. the default polySymbol = "&", but could be a different symbol
     # such as "," or "\"... it can only be *one* symbol, though

# all of this machinery should function just fine in minCharChange
# again, by default polySymbol = "&" (included anyway here for kicks)
minCharChange(tree,
        trait = charPoly,
        polySymbol = "&")
```

`modelMethods` *Model Function Methods: Parameter Names, Bounds and Initial Values*

Description

A large number of functions for obtaining and modifying the parameters of likelihood models made in `paleotree`. These functions allow users to obtain or set parameter names, or obtain and set parameter bounds, both of which are treated as an attribute of the function class used by paleotree. In practice, this allows users to quickly obtain parameter names and upper and lower values for use in bounded optimizers, including reasonable starting values.

Usage

```
parnames(x, ...)

## S3 method for class 'paleotreeFunc'
parnames(x, ...)
```

```
## S3 method for class 'constrained'
parnames(x, ...)

parnames(x) <- value

## S3 replacement method for class 'constrained'
parnames(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parnames(x) <- value

parbounds(x, ...)

## S3 method for class 'paleotreeFunc'
parbounds(x, ...)

## S3 method for class 'constrained'
parbounds(x, ...)

parbounds(x) <- value

## S3 replacement method for class 'constrained'
parbounds(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parbounds(x) <- value

parLower(x, ...)

## S3 method for class 'constrained'
parLower(x, ...)

## S3 method for class 'paleotreeFunc'
parLower(x, ...)

parLower(x) <- value

## S3 replacement method for class 'constrained'
parLower(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parLower(x) <- value

parUpper(x, ...)

## S3 method for class 'constrained'
parUpper(x, ...)
```

```
## S3 method for class 'paleotreeFunc'
parUpper(x, ...)

parUpper(x) <- value

## S3 replacement method for class 'constrained'
parUpper(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parUpper(x) <- value

parInit(x, ...)

## S3 method for class 'constrained'
parInit(x, ...)

## S3 method for class 'paleotreeFunc'
parInit(x, ...)
```

Arguments

x	A function of S3 class 'paleotreeFunc' with all necessary attributes expected of that class, which include parameter names and upper and lower bounds. As I have deliberately not exported the function which creates this class, it should be impossible for regular users to obtain such objects easily without using one of the make functions, which automatically output a function of the appropriate class and attributes.
...	'Ignored arguments to future methods' (i.e. for diversitree). Kept here only so constrainParPaleo is kept as close to the parent method in ddiversitree as possible.
value	The new value with which to replace the parameter names or bounds. Must be a vector of the same length as the number of parameters. For parbounds, must be a list composed of two vectors.

Details

Parameter names cannot be changed for a constrained function.

The parInit function calls the bounds for each parameter and gives a randomly selected value selected from a uniform distribution, using the parameter bounds for each parameter as the bounds on the uniform distribution. This users a shorthand to quickly generate initial parameter values which are within the set bounds, for use in functions such as optim. The random sampling of initial values allows a user to quickly assess if initial parameter values affect the optimization by simply rerunning the function on new values. Infinite initial parameter values (resulting from infinite bounds) are discarded, and replaced with the lower bound value (assuming only upper bounds are infinite...). Some randomly selected initial parameter values may be too high (due to the liberal upper bounds I set for parameters in many of the likelihood functions) and thus users should always try slightly different values to see if the resulting maximum likelihood parameter values change.

As parInit depends on the upper and lower bounds attribute, no function is offered to allow it to be replaced (as there is nothing to replace!).

Value

Returns the sought parameter names, bounds or initial values or (for the replacement methods) returns a modified function with the respective attributes altered.

Author(s)

These functions are strongly based on or inspired by the argnames functions provided for handling models in Rich Fitzjohn's library diversitree, but the functions presented here are derivations written by David Bapst.

See Also

These model methods were introduced to interact with the new model framework introduced in paleotree version >1.9, in particular to interface with constrainParPaleo.

Examples

```
#example with make_durationFreqCont
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r = 0.5)
likFun <- make_durationFreqCont(rangesCont)

#get parameter names
parnames(likFun)

#get the bounds for those parameters
parbounds(likFun)

#can also get these seperately
parLower(likFun)
parUpper(likFun)

#initial parameter values
parInit(likFun)   #arbitrary midway value between par bounds

#can then use these in optimizers, such as optim with L-BFGS-B
#see the example for make_durationFreqCont

#renaming parameter names
likFun2 <- likFun
parnames(likFun2) <- c("extRate","sampRate")
parnames(likFun2)
#test if reset correctly
parnames(likFun2) == c("extRate","sampRate")
#also works for constrained functions
```

```
constrainFun <- constrainParPaleo(likFun,q.1~r.1)
parnames(constrainFun)
#also modified the parameter bounds, see!
parbounds(constrainFun)
parInit(constrainFun)
#but cannot rename parameter for constrained function!
```

`modifyTerminalBranches` *Modify, Drop or Bind Terminal Branches of Various Types (Mainly for Paleontological Phylogenies)*

Description

These functions modify terminal branches or drop certain terminal branches based on various criteria. `dropZLB` drops tip-taxa that are attached to the tree via zero-length terminal branches ("ZLBs"). This is sometimes useful for phylogenies of fossil taxa, as various time-scaling methods often produce these 'ZLBs', taxa whose early appearance causes them to be functionally interpreted as ancestors in some time-scaling methods. Removing 'ZLBs' is advised for analyses of diversification/diversity, as these will appear as simultaneous speciation/extinction events. Note this function only drops tips attached to a terminal zero-length branch; if you want to collapse internal zero-length branches, see the ape function `di2multi`.

Usage

```
dropZLB(tree)

dropExtinct(tree, tol = 0.01, ignore.root.time = FALSE)

dropExtant(tree, tol = 0.01)

addTermBranchLength(tree, addtime = 0.001)

dropPaleoTip(tree, ...)

bindPaleoTip(
  tree,
  tipLabel,
  nodeAttach = NULL,
  tipAge = NULL,
  edgeLength = NULL,
  positionBelow = 0,
  noNegativeEdgeLength = TRUE
)
```

Arguments

tree	A phylogeny, as an object of class phylo. dropPaleoTip requires this input object to also have a tree$root.time element. If not provided for bindPaleoTip, then the $root.time will be presumed to be such that the furthest tip from the root is at time = 0.
tol	Tolerance for determining modern age; used for distinguishing extinct from extant taxa. Tips which end within tol of the furthest distance from the root will be treated as 'extant' taxa for the purpose of keeping or dropping.
ignore.root.time	Ignore tree$root.time in calculating which tips are extinct? tree$root.time will still be adjusted, if the operation alters the tree$root.time.
addtime	Extra amount of time to add to all terminal branch lengths.
...	additional arguments passed to dropPaleoTip are passed to drop.tip.
tipLabel	A character string of length = 1 containing the name of the new tip to be added to tree.
nodeAttach	Node or tip ID number (as given in tree$edge) at which to attach the new tip. See documentation of bind.tip for more details.
tipAge	The age of the tip taxon added to the tree, in time before present (i.e. where present is 0), given in the same units as the edges of the tree are already scaled. Cannot be given if edgeLength is given.
edgeLength	The new edge.length of the terminal branch this tip is connected to. Cannot be given if tipAge is given.
positionBelow	The distance along the edge below the node to be attached to (given in nodeAttach to add the new tip. Cannot be negative or greater than the length of the edge below nodeAttach.
noNegativeEdgeLength	Return an error if a negative terminal edge length is calculated for the new tip.

Details

dropExtinct drops all terminal branches which end before the modern (i.e. extinct taxa). DropExtant drops all terminal branches which end at the modern (i.e. extant/still-living taxa). In both cases, the modern is defined based on tree$root.time if available, or the modern is inferred to be the point in time when the tip furthest from the root (the latest tip) terminates.

If the input tree has a $root.time element, as expected for most phylogeny containing fossil taxa objects handled by this library, that $root.time is adjusted if the relative time of the root divergence changes when terminal branches are dropped. This is typically performed via the function fixRootTime. Adjusted $root.time elements are only given if the input tree has a $root.time element.

addTermBranchLength adds an amount equal to the argument addtime to the terminal branch lengths of the tree. If there is a $root.time element, this is increased by an amount equal to addtime. A negative amount can be input to reduce the length of terminal branches. However, if negative branch lengths are produced, the function fails and a warning is produced. The function addTermBranchLength does *not* call fixRootTime, so the root.time elements in the result tree may be nonsensical, particularly if negative amounts are input.

dropPaleoTip is a wrapper for ape's drop.tip which also modifies the $root.time element if necessary, using fixRootTime. Similarly, bindPaleoTip is a wrapper for phytool's bind.tip which allows tip age as input and modifies the $root.time element if necessary (i.e. if a tip is added to edge leading up to the root).

Note that for bindPaleoTip, tips added below the root are subtracted from any existing $root.edge element, as per behavior of link{bind.tip} and bind.tree. However, bindPaleoTip will append a $root.edge of the appropriate value (i.e., root edge length) if one does not exist (or is not long enough) to avoid an error. After binding is finished, any $root.edge equal to 0 is removed before the resulting tree is output.

Value

Gives back a modified phylogeny as a phylo object, generally with a modified $root.time element.

Author(s)

David W. Bapst. The functions dropTipPaleo and bindTipPaleo are modified imports of drop.tip and bind.tip from packages ape and phytools.

See Also

compareTermBranches, phyloDiv, drop.tip, bind.tip

Examples

```
set.seed(444)
# Simulate some fossil ranges with simFossilRecord
record <- simFossilRecord(
    p = 0.1, q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = 0
    )
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record
    # with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.5)
# Now let's make a tree using taxa2phylo
tree <- taxa2phylo(taxa,obs_time = rangesCont[,2])
# compare the two trees
layout(1:2)
plot(ladderize(tree))
plot(ladderize(dropZLB(tree)))

# reset
layout(1)


# example using dropExtinct and dropExtant
set.seed(444)
```

```
record <- simFossilRecord(
    p = 0.1, q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = c(10,20)
    )
taxa <- fossilRecord2fossilTaxa(record)
tree <- taxa2phylo(taxa)
phyloDiv(tree)
tree1 <- dropExtinct(tree)
phyloDiv(tree1)
tree2 <- dropExtant(tree)
phyloDiv(tree2)


# example using addTermBranchLength
set.seed(444)
treeA <- rtree(10)
treeB <- addTermBranchLength(treeA,1)
compareTermBranches(treeA,treeB)

##########################
# test dropPaleoTip
# (and fixRootTime by extension...)

# simple example
tree <- read.tree(text = "(A:3,(B:2,(C:5,D:3):2):3);")
tree$root.time <- 10
plot(tree, no.margin = FALSE)
axisPhylo()

# now a series of tests, dropping various tips
(test <- dropPaleoTip(tree,"A")$root.time) #  = 7
(test[2] <- dropPaleoTip(tree,"B")$root.time) #  = 10
(test[3] <- dropPaleoTip(tree,"C")$root.time) #  = 10
(test[4] <- dropPaleoTip(tree,"D")$root.time) #  = 10
(test[5] <- dropPaleoTip(tree,c("A","B"))$root.time) #  = 5
(test[6] <- dropPaleoTip(tree,c("B","C"))$root.time) #  = 10
(test[7] <- dropPaleoTip(tree,c("A","C"))$root.time) #  = 7
(test[8] <- dropPaleoTip(tree,c("A","D"))$root.time) #  = 7

# is it all good? if not, fail so paleotree fails...
if(!identical(test,c(7,10,10,10,5,10,7,7))){
     stop("fixRootTime fails!")
     }


##############
# testing bindPaleoTip

# simple example
tree <- read.tree(text = "(A:3,(B:2,(C:5,D:3):2):3);")
tree$root.time <- 20
```

```
plot(tree, no.margin = FALSE)
axisPhylo()

## Not run:

require(phytools)

# bindPaleoTip effectively wraps bind.tip from phytools
# using a conversion like below

tipAge <- 5
node <- 6

# the new tree length (tip to root depth) should be:
# new length = the root time - tipAge - nodeheight(tree,node)

newLength <- tree$root.time-tipAge-nodeheight(tree,node)
tree1 <- bind.tip(tree,
    "tip.label",
    where = node,\
    edge.length = newLength)

layout(1:2)
plot(tree)
axisPhylo()
plot(tree1)
axisPhylo()

# reset
layout(1)


## End(Not run)

# now with bindPaleoTip

tree1 <- bindPaleoTip(tree,"new",nodeAttach = 6,tipAge = 5)

layout(1:2)
plot(tree)
axisPhylo()
plot(tree1)
axisPhylo()

# reset
layout(1)

#then the tip age of "new" should 5
test <- dateNodes(tree1)[which(tree1$tip.label == "new")] == 5
if(!test){
    stop("bindPaleoTip fails!")
    }
```

```
# with positionBelow

tree1 <- bindPaleoTip(
    tree,
    "new",
    nodeAttach = 6,
    tipAge = 5,
    positionBelow = 1
    )

layout(1:2)
plot(tree)
axisPhylo()
plot(tree1)
axisPhylo()

# reset
layout(1)

# at the root

tree1 <- bindPaleoTip(
    tree,
    "new",
    nodeAttach = 5,
    tipAge = 5)

layout(1:2)
plot(tree)
axisPhylo()
plot(tree1)
axisPhylo()

# reset
layout(1)

#then the tip age of "new" should 5
test <- dateNodes(tree1)[which(tree1$tip.label == "new")] == 5
if(!test){
     stop("bindPaleoTip fails!")
     }

# at the root with positionBelow

tree1 <- bindPaleoTip(tree,"new",nodeAttach = 5,tipAge = 5,
positionBelow = 3)

layout(1:2)
plot(tree)
axisPhylo()
plot(tree1)
axisPhylo()
```

```
# reset
layout(1)

#then the tip age of "new" should 5
test <- dateNodes(tree1)[which(tree1$tip.label == "new")] == 5
#and the root age should be 23
test1 <- tree1$root.time == 23
if(!test | !test1){
     stop("bindPaleoTip fails!")
     }
```

---

multiDiv *Calculating Diversity Curves Across Multiple Datasets*

Description

Calculates multiple diversity curves from a list of datasets of taxic ranges and/or phylogenetic trees, for the same intervals, for all the individual datasets. A median curve with 95 percent quantile bounds is also calculated and plotted for each interval.

Usage

```
multiDiv(
  data,
  int.length = 1,
  plot = TRUE,
  split.int = TRUE,
  drop.ZLB = TRUE,
  drop.cryptic = FALSE,
  extant.adjust = 0.01,
  plotLogRich = FALSE,
  yAxisLims = NULL,
  timelims = NULL,
  int.times = NULL,
  plotMultCurves = FALSE,
  multRainbow = TRUE,
  divPalette = NULL,
  divLineType = 1,
  main = NULL
)

plotMultiDiv(
  results,
  plotLogRich = FALSE,
  timelims = NULL,
  yAxisLims = NULL,
  plotMultCurves = FALSE,
```

```
  multRainbow = TRUE,
  divPalette = NULL,
  divLineType = 1,
  main = NULL
)
```

Arguments

data	A list where each element is a dataset, formatted to be input in one of the diversity curve functions listed in DiversityCurves.
int.length	The length of intervals used to make the diversity curve. Ignored if int.times is given.
plot	If TRUE, the median diversity curve is plotted.
split.int	For discrete time data, should calculated/input intervals be split at discrete time interval boundaries? If FALSE, can create apparent artifacts in calculating the diversity curve. See details.
drop.ZLB	If TRUE, zero-length terminal branches are dropped from the input tree for phylogenetic datasets, before calculating standing diversity.
drop.cryptic	If TRUE, cryptic taxa are merged to form one taxon for estimating taxon curves. Only works for objects from simFossilRecord via fossilRecord2fossilTaxa.
extant.adjust	Amount of time to be added to extend start time for (0,0) bins for extant taxa, so that the that 'time interval' does not appear to have an infinitely small width.
plotLogRich	If TRUE, taxic diversity is plotted on log scale.
yAxisLims	Limits for the y (i.e. richness) axis on the plotted diversity curves. Only affects plotting. Given as either NULL (the default) or as a vector of length two as for xlim in the basic R function plot. Time axes will be plotted *exactly* to these values. The minimum value must be more than 1 if plotLogRich = TRUE.
timelims	Limits for the x (time) axis for diversity curve plots. Only affects plotting. Given as either NULL (the default) or as a vector of length two as for xlim in the basic R function plot. Time axes will be plotted *exactly* to these values.
int.times	An optional two-column matrix of the interval start and end times for calculating the diversity curve. If NULL, calculated internally. If given, the argument split.int and int.length are ignored.
plotMultCurves	If TRUE, each individual diversity curve is plotted rather than the median diversity curve and 95 percent quantiles. plotMultCurves = FALSE by default.
multRainbow	If TRUE and plotMultCurves = TRUE, each line is plotted as a different, randomized color using the function rainbow. If FALSE, each line is plotted as a black line. This argument is ignored if divPalette is supplied.
divPalette	Can be used so users can pass a vector of chosen color identifiers for each diversity curve in data which will take precedence over multRainbow. Must be the same length as the number of diversity curves supplied.
divLineType	Used to determine line type (lty) of the diversity curves plotted when plotMultCurves = TRUE. Default is lty = 1 for all curves. Must be either length of 1 or exact length as number of diversity curves.
main	The main label for the figure.
results	The output of a previous run of multiDiv for replotting.

Details

This function is essentially a wrapper for the individual diversity curve functions included in paleotree. `multiDiv` will intuitively decide whether input datasets are continuous-time taxic ranges, discrete-time (binned interval) taxic ranges or phylogenetic trees, as long as they are formatted as required by the respective diversity curve functions. A list that contains a mix of data types is entirely acceptable. A list of matrices output from `fossilRecord2fossilTaxa`, via simulation with `simFossilRecord` is allowable, and treated as input for `taxicDivCont`. Data of an unknown type gives back an error.

The argument `split.int` splits intervals, if and *only* if discrete interval time data is included among the datasets. See the help file for `taxicDivDisc` to see an explanation of why `split.int = TRUE` by default is probably a good thing.

As with many functions in the `paleotree` library, absolute time is always decreasing, i.e. the present day is zero.

The 'averaged' curve is actually the median rather than the mean as diversity counts are often highly skewed (in this author's experience).

The shaded certainty region around the median curve is the two-tailed 95 percent lower and upper quantiles, calculated from the observed data. It is not a true probabilisitic confidence interval, as it has no relationship to the standard error.

Value

A list composed of three elements will be invisibly returned:

`int.times`	A two column matrix giving interval start and end times
`div`	A matrix of measured diversities in particular intervals by rows, with each column representing a different dataset included in the input
`median.curve`	A three column matrix, where the first column is the calculated median curve and the second and third columns are the 95 percent quantile upper and lower bounds

See Also

The diversity curve functions used include: `phyloDiv`, `taxicDivCont` and `taxicDivDisc`.

Also see the function `LTT.average.root` in the package `TreeSim`, which calculates an average LTT curve for multiple phylogenies, the functions `mltt.plot` in ape and `ltt` in `phytools`.

Examples

```
# let's look at this function
    # with some birth-death simulations

set.seed(444)

# multiDiv can take output from simFossilRecord
    # via fossilRecord2fossilTaxa

# what do many simulations run under some set of
    # conditions 'look' like on average?
```

```
set.seed(444)
records <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 10,
    totalTime = 30,
    plot = TRUE
    )

taxa <- lapply(records, fossilRecord2fossilTaxa)

multiDiv(taxa)
# increasing cone of diversity!

# Its even better on a log scale:
multiDiv(taxa, plotLogRich = TRUE)

#######################################
# pure-birth example with simFossilRecord
# note that conditioning is tricky

set.seed(444)
recordsPB <- simFossilRecord(
    p = 0.1,
    q = 0,
    nruns = 10,
    totalTime = 30,
    plot = TRUE
    )

taxaPB <- lapply(recordsPB, fossilRecord2fossilTaxa)
multiDiv(taxaPB, plotLogRich = TRUE)

#compare many discrete diversity curves
discreteRanges <- lapply(taxaPB, function(x)
    binTimeData(
        sampleRanges(x,
            r = 0.5,
            min.taxa = 1
            ),
        int.length = 7)
    )

multiDiv(discreteRanges)

########################################
# plotting a multi-diversity curve for
   # a sample of stochastic dated trees

record <- simFossilRecord(
    p = 0.1, q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
```

```
    nExtant = 0)

taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa, r = 0.5)
rangesDisc <- binTimeData(rangesCont,
    int.length = 1)
# get the cladogram
cladogram <- taxa2cladogram(taxa, plot = TRUE)

#using multiDiv with samples of trees
ttrees <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    randres = TRUE,
    ntrees = 10,
    add.term = TRUE
    )

multiDiv(ttrees)

# uncertainty in diversity history is solely due to
    # the random resolution of polytomies

##########################################################

#using multiDiv to compare very different data types:
   # continuous ranges, discrete ranges, dated tree

# get a single dated tree
ttree <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    add.term = TRUE,
    plot = FALSE
    )

# put them altogether in a list
input <- list(rangesCont, rangesDisc, ttree)

multiDiv(input, plot = TRUE)

# what happens if we use fixed interval times?
multiDiv(input,
    int.times = rangesDisc[[1]],
    plot = TRUE)

layout(1)
```

`nearestNeighborDist` *Nearest Neighbor Distances for Morphological Disparity Studies*

Description

This is a simple function for obtaining nearest neighbor distance from a symmetric pair-wises distance matrix, assumed here to be dissimilarities between pairs of taxa. Per-species NND is returned rather than a mean or other summary value.

Usage

```
nearestNeighborDist(distMat)
```

Arguments

`distMat`	A symmetric, square pair-wise distance matrix, assumed to be a dissimilarity matrix with a diagonal that is either zero values, `NA` values, or a mixture of both. Can be a 'dist' object rather than a numerical matrix. Taxon labels can be applied to the rows and columns (or as labels if a 'dist' object) and will be used to name the resulting output.

Details

This function is mainly included here for pedagogical (teaching) purposes. NND is so simple to calculate, users are urged to write their own functions for primary research purposes.

Typically, the *mean* NND for a group is reported and used to compare different groupings of taxa (such as different time intervals, or different clades). Bootstrapping should be used to generate confidence intervals.

Value

Returns a vector of the nearest neighbor distance for each unit (taxon) in the pair-wise distance matrix, named with the labels from the input distance matrix.

Author(s)

David W. Bapst

References

Bapst, D. W., P. C. Bullock, M. J. Melchin, H. D. Sheets, and C. E. Mitchell. 2012. Graptoloid diversity and disparity became decoupled during the Ordovician mass extinction. *Proceedings of the National Academy of Sciences* 109(9):3428-3433.

Ciampaglio, C. N., M. Kemp, and D. W. McShea. 2001. Detecting changes in morphospace occupation patterns in the fossil record: characterization and analysis of measures of disparity. *Paleobiology* 27(4):695-715.

Foote, M. 1990. Nearest-neighbor analysis of trilobite morphospace. *Systematic Zoology* 39:371-382.

See Also

For the example dataset used in examples, see graptDisparity

Examples

```
#example using graptolite disparity data from Bapst et al. 2012

#load data
data(graptDisparity)

#calculate mean NND
NND <- nearestNeighborDist(graptDistMat)
mean(NND)

#calculate NND for different groups

#group (clade/paraclade) coding
groupID <- graptCharMatrix[,54]+1

groupNND <- numeric(7)
names(groupNND) <- c("Normalo.","Monogr.","Climaco.",
   "Dicrano.","Lasiogr.","Diplogr.","Retiol.")
for(i in unique(groupID)){
   groupNND[i] <- mean(nearestNeighborDist(
      graptDistMat[groupID == i,groupID == i]))
   }
groupNND

#the paraphyletic Normalograptids that survived the HME are most clustered
   #but this looks at all the species at once
   #and doesn't look for the nearest *co-extant* neighbor!
   #need to bring in temporal info to test that
```

---

nodeDates2branchLengths

Obtaining Edge Lengths for Undated Phylogenies Using Known Branching Node and Tip Ages

---

Description

This function takes some undated phylogenetic topology, a set of ages in absolute time, for the internal nodes and (by default) the terminal tips of that phylogeny, and returns a dated phylogeny consistent with those input ages.

Usage

```
nodeDates2branchLengths(nodeDates, tree, allTipsModern = FALSE)
```

Arguments

nodeDates	Under default allTipsModern = FALSE conditions, nodeDates should be a vector of length Ntip(tree) + Nnode(tree) which contains the dates for all terminal tip nodes and internal nodes for the tree, in that order, as numbered in the tree$edge matrix. Such a vector is produced as output by dateNodes. If allTipsModern = TRUE, then the vector should only be as long as the number of nodes, and contain the dates only for those same internal nodes in tree. These dates should always on a descending scale (i.e. time before present), with respect to an absolute time-scale. It is possible for the time 0 date to represent a date far in the future from the latest tip.
tree	An undated phylogeny object, of class phylo, lacking edge lengths. If the tree appears to be dated (i.e. has edge lengths), the function will issue a warning.
allTipsModern	A logical, default is FALSE. If FALSE, then the function expects nodeDates to contain ages for all 'nodes' - both internal branching nodes and terminal tips. If TRUE, then the function will expect nodeDates to contain ages only for internal branching nodes, and all tips will be assumed to be at time 0. (Thus, if your tree is ultrametric but tips aren't all at the modern, do *not* use allTipsModern = TRUE).

Details

The function compute.brtime in package ape does a very similar functionality, but is limited in its application for only ultrametric trees, as it does not allow for tips to have incongruent ages. It also only accepts node ages as on the relative scale where the latest tips are at zero, as assumed in general elsewhere in package ape.

Value

A dated tree as a list of class phylo, with a $root.time element for referencing the tree against absolute time.

Author(s)

David W. Bapst

See Also

This function will likely often be used in conjunction with dateNodes, such as for summarizing node and tip age estimates from a sample of trees, to produce a single dated tree to act as a point estimate. Beware however that point estimates of tree samples may have little resemblance to any individual tree in that sample.

This function should perform identically for ultrametric trees as package ape's function compute.brtime.

Examples

```
set.seed(444)

# we'll do a number of tests, let's check at the end that all are TRUE
```

```
tests <- logical()

# with a non-ultrametric tree
chrono <- rtree(10)
# make an undated tree
notChrono <- chrono
notChrono$edge.length <- NULL

# now lets try with dateNodes in paleotree
nodeTimes <- dateNodes(chrono)
# need to use allTipsModern = FALSE because tip ages are included
chronoRedux <-  nodeDates2branchLengths(tree = notChrono,
    nodeDates = nodeTimes, allTipsModern = FALSE)
# test that its the same
(tests <- c(tests,all.equal.numeric(chrono$edge.length,chronoRedux$edge.length)))

#####################################
# modern ultrametric tree
chrono <- rcoal(10)
# make an undated tree
notChrono <- chrono
notChrono$edge.length <- NULL

# with ultrametric trees, you could just use ape's compute.brtime

# getting branching times with ape
branchingTimes <- branching.times(chrono)
# setting those branching times with ape
chronoRedux <-  compute.brtime(notChrono, branchingTimes)
# test that its the same
(tests <- c(tests,all.equal.numeric(chrono$edge.length,chronoRedux$edge.length)))

# lets do the same thing but with nodeDates2branchLengths

# can use branching.times from ape
# (but only for ultrametric trees!)
chronoRedux <-  nodeDates2branchLengths(tree = notChrono,
    nodeDates = branchingTimes, allTipsModern = TRUE)
# test that its the same
(tests <- c(tests,all.equal.numeric(chrono$edge.length,chronoRedux$edge.length)))

# now lets try with dateNodes in paleotree
nodeTimes <- dateNodes(chrono)
# need to use allTipsModern = FALSE because tip ages are included
chronoRedux <-  nodeDates2branchLengths(tree = notChrono,
    nodeDates = nodeTimes, allTipsModern = FALSE)
# test that its the same
(tests <- c(tests,all.equal.numeric(chrono$edge.length,chronoRedux$edge.length)))

# get just the node times (remove tip dates)
nodeOnlyTimes <- nodeTimes[-(1:Ntip(chrono))]
# let's use the allTipsModern = TRUE setting
chronoRedux <-  nodeDates2branchLengths(tree = notChrono,
```

```
    nodeDates = nodeOnlyTimes, allTipsModern = TRUE)
# test that its the same
(tests <- c(tests,all.equal.numeric(chrono$edge.length,chronoRedux$edge.length)))

# did all tests come out as TRUE?
if(!all(tests)){stop("nodeDates2branchLengths isn't functioning correctly")}
```

`obtainDatedPosteriorTreesMrB` *Get the Sample of Posterior Trees from a Dated Phylogenetic Analysis with MrBayes (Or a Summary Tree, such as the MCCT)*

Description

MrBayes is not great for getting samples of dated posterior phylogenies, or for obtaining certain summary trees from the posterior (specifically the MCCT and MAP, which are specific trees in the posterior). This is because the tree samples as returned are scaled relative to rate parameters in a separate file. This function attempts to automate the handling of multiple files (both *.t* tree files and *.p* parameter files), as well as multiple files associated with separate runs, to obtain samples of posterior trees, or summary trees such as the MCCT or MAP. These resulting trees are now scaled to units of time, but not be placed correctly on an absolute time-scale if all tips are extinct. See details of output below.

Usage

```
obtainDatedPosteriorTreesMrB(
  runFile,
  nRuns = 2,
  burnin = 0.5,
  outputTrees,
  labelPostProb = FALSE,
  getFixedTimes = FALSE,
  getRootAges = FALSE,
  originalNexusFile = NULL,
  file = NULL
)
```

Arguments

`runFile`	A filename in the current directory, or a path to a file that is either a *.p* or *.t* file from a MrBayes analysis. This filename and path will be used for finding additional *.t* and *.p* files, via the `nRuns` settings and assuming that files are in the same directory *and* these files are named under typical MrBayes file naming conventions. (In other words, if you have renamed your *.p* or *.t* files, this function probably won't be able to find them.)

`nRuns` The number of runs in your analysis. This variable is used for figuring out what filenames will be searched for: if you specify that you have less runs than you actually ran in reality, then some runs won't be examined in this function. Conversely, specify too many, and this function will throw an error when it cannot find files it expects but do not exist. The default for this argument (*two* runs) is based on the default number of runs in MrBayes.

`burnin` The fraction of trees sampled in the posterior discarded and not returned by this function directly, nor included in calculation of summary trees. Must be a numeric value greater than 0 and less than 1.

`outputTrees` Determines the output trees produced; for format of output, see section on returned Value below. Must be of length one, and either `"all"`, which means all trees from the post-burn-in posterior will returned, a number greater than zero, which will be the number of trees randomly sampled from across the post-burn-in posterior and returned, or a label for a type of summary tree selected from the posterior based on various properties. The two most commonly seen such point-estimate-summaries are the *MCCT* tree, which stands for the 'maximum clade compatibility tree', and the *MAP* tree, which stands for the 'maximum a posteriori tree'. The MCCT is the single tree from the post-burn-in posterior which contains the set of bifurcations (clades) with the highest product of posterior probabilities (i.e. are found on the most trees in the post-burn-in posterior). The MCCT tree is returned if the argument `outputTrees = "MCCT"` is used. The MAP is the single tree from the post-burn-in posterior with the highest posterior probabilty associated with it. Unfortunately, versions of `paleotree` prior to version 3.2.1 did not use the posterior probability to select the supposed 'MAP' tree. MrBayes provides two values for each sampled tree and corresponding parameters: `LnPr`, the log prior probability of the current parameter proposals under the specified prior distributions, and `LnL`, the log likelihood of the cold chain, i.e. the log-likelihood of the sampled parameter values, given the observed data and specified models. Neither of these are the posterior probability. The true posterior probability (as given by Bayes Theorem) is the product of the likelihood and the prior probability, divided by the likelihood of the model, the latter of which is very rarely known. More commonly, the calculable portion of the posterior probability is the product of the likelihood and the prior probability; or, here, easily calculated as the log posterior probability, as the sum of the log likelihood and log prior probability. Given confusion over application of 'MAP' trees in previous version of paleotree, three options are available: `"MAPosteriori"` for the Maximum A Posteriori tree (the MAP tree, or the single tree in the posterior with the highest posterior probability, as given by `LnPr + LnL`), `"MAPriori"` for the Maximum A Priori tree (the tree in the posterior sample with the highest prior probability, indep of the data), and `"MaxLikelihood"`, the tree with the highest model likelihood given the data, ignoring the prior probability. The former option of `outputTrees = "MAP"` is deprecated, as its previous implementation only examine `LnPr` and thus returned the tree now referred to here as the `"MAPriori"` tree. Interestingly, this bug had no effect when tip-dating methods is applied to datasets with no character matrix is provided (an empty matrix of '?' missing values is used) in order to find dated phylogenies that maximize the fit to the dated tree prior, as the log likelihood for a tree with an empty matrix is *always* zero, and thus the posterior probability is

always exactly identical to the prior probability. Overall, the Maximum A Posteriori tree is the "best" tree based on the metric most directly considered by Bayesian analysis for proposal acceptance, but the MCCT may be the best tree for summarizing topological support. In either case, point estimates of topology are often problematic summaries for phylogenetic analyses.

`labelPostProb` Logical. If `TRUE`, then nodes of the output tree will be labeled with their respective posterior probabilities, as calculated based on the frequency of a clade occurring across the post-burn-in posterior tree sample. If `FALSE`, this is skipped.

`getFixedTimes` If `TRUE`, this function will also look for, scan, and parse an associated NEXUS file. Ignoring any commented lines (i.e., anything between a set of rectangular brackets `[]`), commands for fixing taxa will be identified, parsed and returned to the user, either as a message printed to the R console if output is written to a file, or as a attribute named 'fixed ages' if output as an R object (formatted as a two-column table of OTU names and their respective fixed ages).

Please note: the code for `getFixedTimes = TRUE` contains a `while()` loop in it for removing nested series of square brackets (i.e. treated as comments in NEXUS files). Thus files with ridiculously nested series of brackets may cause this code to take a while to complete, or may even cause it to hang.

`getRootAges` `FALSE` by default. If `TRUE`, and `getFixedTimes = TRUE` as well as `file = NULL` (such that trees will be assigned within the R memory rather than saved to an external file), the functions `setRootAge` and its wrapper function `setRootAges` will be applied to the output so that all output trees have `root.time` elements for use with other functions in `paleotree` as well as other packages.

`originalNexusFile`

Filename (and possibly path too) to the original NEXUS file for this analysis. Only tried if `getFixedTimes = TRUE`. If `NULL` (the default), then this function will instead try to find a NEXUS file with the same name as implied by the filename used in other inputs. If this file cannot be found, the function will fail.

`file` Filename (possibly with path) as a character string leading to a file which will be overwritten with the output trees (or summary tree), as a NEXUS file. If `NULL` (the default), the output will instead be directly returned by this function.

Details

This function is most useful for dealing with dating analyses in MrBayes, particularly when tip-dating a tree with fossil taxa, as the half-compatibility and all-compatibility summary trees offered by the '`sumt`' command in MrBayes can have issues properly portraying summary trees from such datasets.

Value

Depending on argument `file`, the output tree or trees is either returned directly, or instead written out in NEXUS format via ape's `write.NEXUS` function to an external file. The output will consist either of multiple trees sampled from the post-burn-in posterior, or will consist of a single phylogeny (a summary tree, either the MCCT or the MAP - see the details for the argument `outputTrees`).

If the argument `setRootAges = TRUE` is not used, users are warned that the resulting dated trees will not have `$root.time` elements necessary for comparison against an absolute time-scale. Wile

the trees may be scaled to units of absolute time now, rather than with branch lengths expressed in the rate of character change, the dates estimated by some phylogenetics functions in R may give inaccurate estimates of when events occur on the absolute time-scale if all tips are extinct. This is because most functions for phylogenetics in R (and elsewhere) will instead presume that the latest tip will be at time 0 (the modern), which may be wrong if you are using `paleotree` for analyzing *paleontological* datasets consisting of entirely extinct taxa. This can be solved by using argument `getFixedTimes = TRUE` to obtain fixed tip ages, and then scaling the resulting output to absolute time using the argument `setRootAges = TRUE`, which obtains a `$root.time` element for each tree using the functions `setRootAge` and `setRootAges` (for single and multiple phylogenies).

Author(s)

David Bapst, with rescaling of raw output trees via code originally written by Nicholas Crouch.

See Also

When the arguments `getFixedTimes = TRUE` and `setRootAges = TRUE` are used, the resulting output will be scaled to absolute time with the available fixed ages using functions `setRootAge` and `setRootAges` (for single and multiple phylogenies). This is only done if fixed ages are available and if the tree is not being saved to an external file.

Maximum Clade Credibility trees are estimated using the function `maxCladeCred` in package phangorn.

See function `link{tipDatingCompatabilitySummaryMrB}` for additional ways of solely evaluating the topoligical information in trees taken from MrBayes posterior samples.

Examples

```
## Not run:

MCCT <- obtainDatedPosteriorTreesMrB(
  runFile = "C:\\myTipDatingAnalysis\\MrB_run_fossil_05-10-17.nex.run1.t",
  nRuns = 2,
  burnin = 0.5,
  outputTrees = "MCCT",
  file = NULL)

MAP <- obtainDatedPosteriorTreesMrB(
  runFile = "C:\\myTipDatingAnalysis\\MrB_run_fossil_05-10-17.nex.run1.t",
  nRuns = 2,
  burnin = 0.5,
  getFixedTimes = TRUE,
  outputTrees = "MAPosteriori",
  file = NULL)

# get a root age from the fixed ages for tips
setRootAge(tree = MAP)

#pull a hundred trees randomly from the posterior
hundredRandomlySelectedTrees <- obtainDatedPosteriorTreesMrB(
  runFile = "C:\\myTipDatingAnalysis\\MrB_run_fossil_05-10-17.nex.run1.t",
```

```
  nRuns = 2,
  burnin = 0.5,
  getFixedTimes = TRUE,
  getRootAges = TRUE,
  outputTrees = 100,
  file = NULL)


## End(Not run)
```

`occData2timeList` *Converting Occurrences Data to a* `timeList` *Data Object*

Description

This function converts occurrence data, given as a list where each element is a different taxon's occurrence table (containing minimum and maximum ages for each occurrence), to the `timeList` format, consisting of a list composed of a matrix of lower and upper age bounds for intervals, and a second matrix recording the interval in which taxa first and last occur in the given dataset.

Usage

```
occData2timeList(occList, intervalType = "dateRange")
```

Arguments

`occList`	A list where every element is a table of occurrence data for a different taxon, such as that returned by `taxonSortPBDBocc`. The occurrence data can be either a two-column matrix composed of the lower and upper age bounds on each taxon occurrence, or has two named variables which match any of the field names given by the PBDB API under either the `'pbdb'` vocab or `'com'` (compact) vocab for early and late age bounds.
`intervalType`	Must be either `"dateRange"` (the default), `"occRange"` or `"zoneOverlap"`. Please see details below.

Details

This function should translate taxon-sorted occurrence data, which could be Paleobiology Database datasets sorted by `taxonSortPBDBocc` or any data object where occurrence data (i.e. age bounds for each occurrence) for different taxa is separated into different elements of a named list.

The Usage of the Argument `intervalType`:

The argument `intervalType` controls the algorithm used for obtain first and last interval bounds for each taxon, of which there are several options for `intervalType` to select from:

"dateRange" The default option. The bounds on the first appearances are the span between the oldest upper and lower bounds of the occurrences, and the bounds on the last appearances are the span between the youngest upper and lower bounds across all occurrences. This is guaranteed to provide the smallest bounds on the first and last appearances, and was originally suggested to the author by J. Marcot.

"occRange" This option returns the smallest bounds among (a) the oldest occurrences for the first appearance (i.e. all occurrences with their lowest bound at the oldest lower age bound), and (b) the youngest occurrences for the last appearance (i.e. all occurrences with their uppermost bound at the youngest upper age bound).

"zoneOverlap" This option is an attempt to mimic the stratigraphic range algorithm used by PBDB Classic which "finds the oldest base that is older than at least part of all the intervals and the youngest that is younger than at least part of all the intervals" (personal communication, J. Alroy). This is a somewhat more complex case as we are trying to obtain a timeList object. So, for calculating the bounds of the first interval a taxon occurs in, the zoneOverlap algorithm looks for all occurrences that overlap with the age range of the earliest-most occurrence and (1) obtains their earliest boundary ages and returns the latest-most earliest age boundary among these overlapping occurrences and (2) obtains their latest boundary ages and returns the earliest-most latest age boundary among these overlapping occurrences. Similarly, for calculating the bound of the last interval a taxon occurs in, the zoneOverlap algorithm looks for all occurrences that overlap with the age range of the latest-most occurrence and (1) obtains their earliest boundary ages and returns the latest-most earliest age boundary among these overlapping occurrences and (2) obtains their latest boundary ages and returns the earliest-most latest age boundary among these overlapping occurrences.
On theoretical grounds, one could probably describe the zone-of-overlap algorithm as minimizing taxonomic age ranges by assuming that all overlapping occurrences at the start and end of a taxon's range probably describe a very similar first and last appearance (FADs and LADs), and thus picks the occurrence with bounds that extends the taxonomic range the least. However, this does come with a downside that if these occurrences are not essentially repeated attempts to capture the same FAD or LAD, then the zone-of-overlap algorithm is not an accurate depiction of the uncertainty in the ages. The true biological range of a taxon might be well outside the bounds obtained using the zone-of-overlap algorithm. A more conservative approach is the "dateRange" algorithm which finds the smallest possible bounds on the endpoints of a taxon's range without ignoring uncertainty from any particular set of occurrences.

Value

Returns a standard timeList data object, as used by many other paleotree functions, like bin_timePaleoPhy, bin_cal3TimePaleoPhy and taxicDivDisc

Author(s)

David W. Bapst, with the "dateRange" algorithm suggested by Jon Marcot.

See Also

Occurrence data as commonly used with paleotree functions can be obtained with link{getPBDBocc}, and sorted into taxa by taxonSortPBDBocc, and further explored with this function and plotOccData. Also, see the example graptolite dataset at graptPBDB

Examples

```
data(graptPBDB)

graptOccSpecies <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "species",
   onlyFormal = FALSE)
graptTimeSpecies <- occData2timeList(occList = graptOccSpecies)

head(graptTimeSpecies[[1]])
head(graptTimeSpecies[[2]])

graptOccGenus <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "genus",
   onlyFormal = FALSE
   )
graptTimeGenus <- occData2timeList(occList = graptOccGenus)

layout(1:2)
taxicDivDisc(graptTimeSpecies)
taxicDivDisc(graptTimeGenus)

# the default interval calculation is "dateRange"
# let's compare to the other option, "occRange"
   # but now for graptolite *species*

graptOccRange <- occData2timeList(
   occList = graptOccSpecies,
   intervalType = "occRange"
   )

#we would expect no change in the diversity curve
   #because there are only changes in th
       #earliest bound for the FAD
       #latest bound for the LAD
#so if we are depicting ranges within maximal bounds
   #dateRanges has no effect
layout(1:2)
taxicDivDisc(graptTimeSpecies)
taxicDivDisc(graptOccRange)
#yep, identical!

#so how much uncertainty was gained by using dateRange?

# write a function for getting uncertainty in first and last
   # appearance dates from a timeList object
sumAgeUncert <- function(timeList){
   fourDate <- timeList2fourDate(timeList)
   perOcc <- (fourDate[,1] - fourDate[,2]) +
       (fourDate[,3] - fourDate[,4])
   sum(perOcc)
```

```
  }

#total amount of uncertainty in occRange dataset
sumAgeUncert(graptOccRange)
#total amount of uncertainty in dateRange dataset
sumAgeUncert(graptTimeSpecies)
#the difference
sumAgeUncert(graptOccRange) - sumAgeUncert(graptTimeSpecies)
#as a proportion
1 - (sumAgeUncert(graptTimeSpecies) / sumAgeUncert(graptOccRange))

#a different way of doing it
dateChange <- timeList2fourDate(graptTimeSpecies) -
    timeList2fourDate(graptOccRange)
apply(dateChange, 2, sum)
#total amount of uncertainty removed by dateRange algorithm
sum(abs(dateChange))

layout(1)
```

`optimPaleo` *Simplified Optimizer for* `paleotree` *Likelihood Functions*

Description

This function is a deliberately simplistic automation wrapper for the function `optim` and the use of the `"L-BFGS-B"` optimizing method, with initial parameter values and bounds provided with `parInit`, `parLower` and `parUpper`. It is mainly provided here as a shorthand to be used in educational demonstrations where model-fitting is not the primary focus, and use in actual analyses should be avoided.

Usage

```
optimPaleo(modelFun)
```

Arguments

`modelFun`	A likelihood function for a model, of class `paleotreeFunc`.

Details

This is mainly provided in this publicly released package for pedagogical reasons. Users seeking an optimizer for their own analytical purposes should write their own `optim` function.

Value

Returns the results from using `optim`.

See Also

constrainParPaleo and modelMethods

Examples

```
# This function simply replicates optim() as shown below
    # where modelFun is the likelihood function

#optim(parInit(modelFun),modelFun,
# lower = parLower(modelFun),upper = parUpper(modelFun),
# method = "L-BFGS-B",control = list(maxit = 1000000))
```

---

parentChild2taxonTree *Create a Taxonomy-Based Phylogeny ('Taxon Tree') from a Table of Parent-Child Taxon Relationships*

Description

This function takes a two-column matrix of taxon names, indicating a set of binary parent-taxon:child-taxon paired relationships with a common root, and returns a 'taxonomy-tree' phylogeny object of class phylo.

Usage

```
parentChild2taxonTree(parentChild, tipSet = "nonParents", cleanTree = TRUE)
```

Arguments

parentChild	A two-column matrix of type character where each element is a taxon name. Each row represents a parent-child relationship with first the parent (column 1) taxon name and then the child (column 2).
tipSet	This argument controls which taxa are selected as tip taxa for the output tree. The default tipSet = "nonParents" selects all child taxa which are not listed as parents in parentChild. Alternatively, tipSet = "all" will add a tip to every internal node with the parent-taxon name encapsulated in parentheses.
cleanTree	When TRUE (the default), the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If FALSE, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.

Details

All taxa listed must be traceable via their parent-child relationships to a single, common ancestor which will act as the root node for output phylogeny. Additionally, the root used will be the parent taxon to all tip taxa closest in terms of parent-child relationships to the tip taxa: i.e., the most recent common ancestor. Ancestral taxa which are singular internal nodes that trace to this root are removed, and a message is printed.

Value

A phylogeny of class `phylo`, with tip taxa as controlled by argument `tipSet`. The output tree is returned with no edge lengths.

The names of higher taxa than the tips should be appended as the element `$node.label` for the internal nodes.

Author(s)

David W. Bapst

See Also

`makePBDBtaxonTree`, `taxonTable2taxonTree`

Examples

```
#let's create a small, really cheesy example
pokexample <- rbind(
    cbind("Squirtadae", c("Squirtle","Blastoise","Wartortle")),
    c("Shelloidea","Lapras"), c("Shelloidea","Squirtadae"),
    c("Pokezooa","Shelloidea"), c("Pokezooa","Parasect"),
    c("Rodentapokemorpha","Linoone"), c("Rodentapokemorpha","Sandshrew"),
    c("Rodentapokemorpha","Pikachu"), c("Hirsutamona","Ursaring"),
    c("Hirsutamona","Rodentapokemorpha"), c("Pokezooa","Hirsutamona")
    )

#Default: tipSet = 'nonParents'
pokeTree <- parentChild2taxonTree(
    parentChild = pokexample,
    tipSet = "nonParents")
plot(pokeTree)
nodelabels(pokeTree$node.label)

#Get ALL taxa as tips with tipSet = 'all'
pokeTree <- parentChild2taxonTree(
    parentChild = pokexample,
    tipSet = "all")
plot(pokeTree)
nodelabels(pokeTree$node.label)


## Not run:

# let's try a dataset where not all the
    # taxon relationships lead to a common root

pokexample_bad <- rbind(
    cbind("Squirtadae", c("Squirtle","Blastoise","Wartortle")),
    c("Shelloidea","Lapras"), c("Shelloidea","Squirtadae"),
    c("Pokezooa","Shelloidea"), c("Pokezooa","Parasect"),
```

```
    c("Rodentapokemorpha","Linoone"), c("Rodentapokemorpha","Sandshrew"),
    c("Rodentapokemorpha","Pikachu"), c("Hirsutamona","Ursaring"),
    c("Hirsutamona","Rodentapokemorpha"), c("Pokezooa","Hirsutamona"),
    c("Umbrarcheota","Gengar")
    )

# this should return an error
    # as Gengar doesn't share common root
pokeTree <- parentChild2taxonTree(parentChild = pokexample_bad)


# another example, where a taxon is listed as both parent and child
pokexample_bad2 <- rbind(
    cbind("Squirtadae", c("Squirtle","Blastoise","Wartortle")),
    c("Shelloidea", c("Lapras","Squirtadae","Shelloidea")),
    c("Pokezooa","Shelloidea"), c("Pokezooa","Parasect"),
    c("Rodentapokemorpha","Linoone"), c("Rodentapokemorpha","Sandshrew"),
    c("Rodentapokemorpha","Pikachu"), c("Hirsutamona","Ursaring"),
    c("Hirsutamona","Rodentapokemorpha"), c("Pokezooa","Hirsutamona"),
    c("Umbrarcheota","Gengar")
    )

#this should return an error, as Shelloidea is its own parent
pokeTree <- parentChild2taxonTree(parentChild = pokexample_bad2)


## End(Not run)



# note that we should even be able to do this
    # with ancestor-descendent pairs from
    # simulated datasets from simFossilRecord, like so:
set.seed(444)
record <- simFossilRecord(
    p = 0.1, q = 0.1, nruns = 1,
    nTotalTaxa = c(30, 40),
    nExtant = 0
    )
taxa <- fossilRecord2fossilTaxa(record)
# need to reorder the columns so parents
    # (ancestors) first, then children
parentChild2taxonTree(parentChild = taxa[,2:1])
# now note that it issues a warning that
    # the input wasn't type character
    # and it will be coerced to be such
```

---

perCapitaRates *Instantaneous* per-Capita *Rates of Origination and Extinction from the Fossil Record*

Description

Calculates and plots per-capita origination and extinction rates from sequential discrete-time taxon ranges, following Foote (2000).

Usage

```
perCapitaRates(
  timeList,
  plot = TRUE,
  logRates = FALSE,
  drop.extant = FALSE,
  isExtant = NULL,
  jitter = TRUE,
  legendPosition = "topleft"
)
```

Arguments

timeList	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. See details.
plot	If TRUE, the per-capita origination and extinctions rates are plotted for each interval. Rates which cannot be calculated for an interval will not be plotted, thus appearing as a gap in the plotted graph. The author takes no responsibility for the aesthetics of this plot.
logRates	If TRUE, rates plotted on log scale.
drop.extant	Drops all extant taxa from a dataset before calculating per-capita origination and extinction rates.
isExtant	A vector of TRUE and FALSE values, same length as the number of taxa in the second matrix of timeList, where TRUE values indicate taxa that are alive in the modern day (and thus are boundary crossers which leave the most recent interval). By default, this argument is NULL and instead which taxa are extant is inferred based on which taxa occur in an interval with start and end times both equal to zero. See details.
jitter	If TRUE (default) the extinction rate will be plotted slightly ahead of the origination rate on the time axis, so the two can be differentiated.
legendPosition	The position of a legend indicating which line is origination rate and which is extinction rate on the resulting plot. This is given as the possible positions for argument x of the function legend, and by default is "topleft", which will be generally useful if origination and extinction rates are initially low. If legendPosition = NA, then a legend will not be plotted.

Details

This function calculates the per-capita rates of taxonomic origination and extinction from paleontological range data, as described by Foote (2000). These values are the instantaneous rate of either type of event occurring per lineage time-units. Although Foote (2001) also presents a number of alternative rates collected from the prior literature such as the 'Van Valen' rate metrics, these are not

implemented here, but could be estimated using the matrix invisibly output by this function (See Foote, 2000, for the relevant equations for calculating these).

The `timeList` object should be a list composed of two matrices, the first matrix giving by-interval start and end times (in absolute time), the second matrix giving the by-taxon first and last appearances in the intervals defined in the first matrix, numbered as the rows. Absolute time should be decreasing, while the intervals should be numbered so that the number increases with time. Taxa alive in the modern should be either (a) listed in `isExtant` or (b) listed as last occurring in a time interval that begins at time 0 and ends at time 0. See the documentation for the time-scaling function `bin_timePaleoPhy` and the simulation function `binTimeData` for more information on formatting.

Unlike some functions in `paleotree`, such as the diversity curve functions, intervals must be both sequential and non-overlapping. The diversity curve functions deal with such issues by assuming taxa occur from the base of the interval they are first found in until the end of the last interval they are occur in. This inflation of boundary crossers could badly bias estimates of per-capita diversification rates.

Value

This function will invisibly return a ten column matrix, where the number of rows is equal to the number of intervals. The first two columns are interval start and end times and the third column is interval length. The fourth through eighth column is the four fundamental classes of taxa from Foote (2001): `Nbt`, `NbL`, `NFt`, `NFL` and their sum, `N`. The final two columns are the per-capita rates estimated for each interval in units per lineage time-units; the ninth column is the origination rate (`pRate`) and the tenth column is the extinction rate (`qRate`).

References

Foote, M. 2000 Origination and extinction components of taxonomic diversity: general problems. Pp. 74–102. In D. H. Erwin, and S. L. Wing, eds. Deep Time: Paleobiology's Perspective. The Paleontological Society, Lawrence, Kansas.

See Also

`DiversityCurves`, `SamplingConv`

Examples

```
#with the retiolinae dataset
data(retiolitinae)
perCapitaRates(retioRanges)

# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(
    p = 0.1, q = 0.1, nruns = 1, nTotalTaxa = c(80,100), nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)

#simulate a fossil record with imperfect sampling with sampleRanges()
```

```
rangesCont <- sampleRanges(taxa,r = 0.5)

#Now let's use binTimeData() to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length = 5)

#and get the per-capita rates
perCapitaRates(rangesDisc)

#on a log scale
perCapitaRates(rangesDisc,logRates = TRUE)


#get mean and median per-capita rates
res <- perCapitaRates(rangesDisc,plot = FALSE)

apply(res[,c("pRate","qRate")],2,mean,na.rm = TRUE)

apply(res[,c("pRate","qRate")],2,median,na.rm = TRUE)

##############################
#with modern taxa
set.seed(444)

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
    nExtant = c(10,50)
    )

taxa <- fossilRecord2fossilTaxa(record)

#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r = 0.5,,modern.samp.prob = 1)

#Now let's use binTimeData() to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length = 5)

#and now get per-capita rates
perCapitaRates(rangesDisc)
```

`perfectParsCharTree` *Simulate a Set of Parsimony-Informative Characters for a Phylogeny*

Description

Creates a simulated set of parsimony-informative characters for a given rooted phylogeny, with characters shared out equally across nodes in the phylogeny, with any remaining characters assigned randomly to nodes.

Usage

```
perfectParsCharTree(tree, nchar)
```

Arguments

tree	A phylogeny, as an object of class phylo.
nchar	Number of parsimonious binary characters to simulate on the phylogeny.

Details

This function takes some a tree and places a number of binary characters on the tree, with character states arranged as if the derived condition was gained once, at a single node, and never lost. This ensures that the resulting simulated character matrices have no character conflict, supporting a single solution under maximum parsimony.

If nchar is greater than the number of nodes on the input phylogeny (ignoring the root), then characters are first placed to evenly cover all nodes, with as many full passes of tree as possible. Any characters in excess are placed at random nodes, without replacement. In other words, if a tree has 10 nodes (plus the root) and 25 characters are simulated, 20 of those characters will consist of two 10-character 'full passes' of the tree. The remaining five will be randomly dropped on the tree.

If few characters are simulated than the number of nodes, these are randomly placed on the given topology without replacement, just as described above.

This function assumes, like almost every function in paleotree, that the tree given is rooted, even if the most basal node is a polytomy.

Value

A matrix of nchar parsimonious binary characters for each taxon on tree, with states 0 and 1.

Author(s)

David W. Bapst

Examples

```
data(retiolitinae)

#fewer characters than nodes
perfectParsCharTree(retioTree,nchar = 10)

#same as number of nodes (minus root)
perfectParsCharTree(retioTree,nchar = 12)

#more characters than the number of nodes
perfectParsCharTree(retioTree,nchar = 20)
```

plotOccData *Plotting Occurrence Data Across Taxa*

Description

plotOccData takes occurrence data which has been sorted into a by-taxon list, such as that output by taxonSortPBDBocc or may be output by simulations using sampleRanges and produces a plot showing the age uncertainty associated with individual occurrences, with occurrences of the same taxon grouped by color.

Usage

```
plotOccData(
  occList,
  groupLabel = NULL,
  occColors = NULL,
  lineWidth = NULL,
  xlims = NULL
)
```

Arguments

occList	A list where every element is a table of occurrence data for a different taxon, such as that returned by taxonSortPBDBocc. The occurrence data can be either a two-column matrix composed of the lower and upper age bounds on each taxon occurrence, or has two named variables which match any of the field names given by the PBDB API under either the 'pbdb' vocab or 'com' (compact) vocab for early and late age bounds.
groupLabel	A character vector with a single string giving the name for the occurrence dataset used, such as the taxonomic name of the group examined. If not given (the default) a generic plot title is appended.
occColors	A vector of numbers or characters indicating colors on a color palette for use with the basic plot function. Must be the same length as occList. If empty, as with the default, the colors used are sampled randomly from the rainbow function.
lineWidth	A numeric value giving the length to be used for the width of lines plotted in plotOccData. If not given (the default), this is calculated using an algorithm that selects an optimal line width for plotting.
xlims	A two element vector controlling the width of the horizontal time-scale the occurrence bars are plotted against. By default, this is not given and calculated internally.

Details

This function was originally conceived of in the following blog post: Link

Value

This function will invisibly return a list, with each per-taxon element containing the two-column matrix of age bounds for occurrences.

Author(s)

David W. Bapst

See Also

Occurrence data as commonly used with `paleotree` functions can be obtained with `link{getPBDBocc}`, and sorted into taxa by `taxonSortPBDBocc`, and further explored with this function and `occData2timeList`. Also, see the example graptolite dataset at `graptPBDB` and the example graptolite dataset at `graptPBDB`

Examples

```
#load example graptolite PBDB occ dataset
data(graptPBDB)

#get formal genera
occSpecies <- taxonSortPBDBocc(graptOccPBDB, rank = "species")

#plot it!
plotOccData(occSpecies)

#this isn't too many occurrences, because there are so few
    #formal grapt species in the PBDB

#genera is messier...

#get formal genera
occGenus <- taxonSortPBDBocc(graptOccPBDB, rank = "genus")

#plot it!
plotOccData(occGenus)

#some of those genera have occurrences with very large
   #age uncertainties on them!
```

---

plotPhyloPicTree *Plot a Phylogeny with Organismal Silhouettes from PhyloPic, Called Via the Paleobiology Database API*

Description

This function will take a phylogeny, preferably a taxonomy-tree created from classification information and/or parent-child taxon information pulled from the Paleobiology Database via function `makePBDBtaxonTree`, and use the Paleobiology Database's API to plot silhouettes of each given tip taxon in replacement of their normal tip labels.

Usage

```
plotPhyloPicTree(
  tree,
  taxaDataPBDB = tree$taxaDataPBDB,
  maxAgeDepth = NULL,
  depthAxisPhylo = FALSE,
  colorAxisPhylo = "black",
  addTaxonStratDurations = FALSE,
  taxaStratRanges = tree$tipTaxonFourDateRanges,
  stratDurationBoxWidth = 0.7,
  sizeScale = 0.9,
  removeSurroundingMargin = TRUE,
  orientation = "rightwards",
  resetGrPar = TRUE,
  taxaColor = NULL,
  transparency = 1,
  cacheDir = "cachedPhyloPicPNGs",
  cacheImage = TRUE,
  noiseThreshold = 0.1,
  rescalePNG = TRUE,
  trimPNG = TRUE,
  colorGradient = "original",
  ...
)
```

Arguments

`tree`	A phylogeny of class `phylo` which will be plotted, with the terminal tip taxa replaced by silhouettes. The tree will be plotted with edge lengths.
`taxaDataPBDB`	A `data.frame` of taxonomic data from the Paleobiology Database containing an `$image_no` variable, as returned when `show = "img"` is used. See *Details*.
`maxAgeDepth`	The maximum tree depth displayed for a tree given with branch lengths (age depth for a dated tree). The portion of the phylogeny older than this date will not be shown. `NULL` by default. If provided, the input tree must have branch lengths in `tree$edge.length`.
`depthAxisPhylo`	If `TRUE`, the ape function `axisPhylo` is run to add an axis of the tree depth to the tree, which must have branch lengths. `FALSE` by default. If `removeSurroundingMargin = TRUE`, which removes extraneous margins, the margins of the plot will be adjusted to make room for the plotted axis.
`colorAxisPhylo`	A color in which the axis for the phylogenetic's depth (generally a time-scale) will be plotted in, for both the axis, its tickmarks, and the labels for the tickmarks.
`addTaxonStratDurations`	If `TRUE`, solid color boxes are plotted on the tree to indicated known taxon ranges, from the oldest possible for the oldest known observation of that taxon, to the youngest possible age for the youngest known observation of that taxon.

This information needs to be supplied as input, see argument taxaStratRanges. If FALSE (the default), nothing happens.

taxaStratRanges

A matrix of four-date range information, as is often used when converting Paleobiology Database taxon data to a dated tree. By default, this is expected to be located at tree$tipTaxonFourDateRanges, which is where such data is placed by default by the function dateTaxonTreePBDB. If addTaxonStratDurations = FALSE (the default), this data is not checked for.

stratDurationBoxWidth

The width of the stratigraphic duration boxes plotted for taxa on the tree. By default, this is 0.7 units. If addTaxonStratDurations = FALSE (the default), this argument affects nothing.

sizeScale The default is sizeScale = 0.9.

removeSurroundingMargin

This argument controls the no.margin argument in the function plot.phylo, which controls whether a (very large) margin is placed around the plotted tree, or not. By default, plotPhyloPicTree will suppress that margin, so that the plotted tree goes (very nearly) to the edges of the plotting area.

orientation Controls the direction the phylogeny is plotted in - can be either "rightwards" or "upwards".

resetGrPar If TRUE (the default), the graphic parameters are reset, so that choices of margins and coordinate system manipulation done as part of this function do not impact the next plot made in this graphic device. If you need to add additional elements to the plot after running this function within R, you should set this argument to FALSE.

taxaColor Controls the color of plotted PhyloPics. Can either be NULL (the default, all taxa will be plotted as black), or a character vector that is either length 1, or the same length as the number of taxa. If taxaColor is length 1, then the value is either interpreted as matching a tip label (in which case, the named taxon will be highlighted in bright red), or as a color, which all PhyloPics will then be plotted as that color. If the vector is the same length as the number of taxa on tree, each value should be a character value of a named color in base R, allowing user control over each PhyloPic individually. All PhyloPics expressed in colors other than the default black are transformed as under the argument colorGradient = "trueMonochrome", so that the PhyloPic is expressed with no intermediate gray-scale values.

transparency A numeric value between 0 and 1, either length 1, or the same length as the number of tips on tree. This indicates the transparency of either all the plotted PhyloPics, or allows user control over each PhyloPic individually. The default is 1, which represents maximum opaqueness, applied to all PhyloPics.

cacheDir If not NULL, this value is used as the name of a sub-directory of the working directory for which to look for (or store) cached versions of PhyloPic PNGs to save on processing speed and the need to pull the images from an external PNG. If NULL, then cached images will not be checked for, and images downloaded will not be cached. The default is

`cacheImage` If `TRUE` (the default), images downloaded from the Paleobiology Database and/or the PhyloPic Database will be cached to save on processing speed and avoid the need to pull the images from an external PNG.

`noiseThreshold` A threshold for noise in the PNG from PhyloPic to be treated as meaningless noise (i.e. a color that is effectively whitespace) and thus can be trimmed as empty margin which can be trimmed before the silhouette is plotted. The units for this argument are on a scale from 0 to 1, with 0 being true white space, and values between 0 and 0.5 representing colors closer to whitespace than true black. The default is `noiseThreshold = 0.1`.

`rescalePNG` If `TRUE` (the default), the downloaded PhyloPic has its color values rebalanced to go from the most extreme white to the most extreme black. Some (especially PBDB's versions) have varying levels of gray compression-related artifacts and may not be properly on a black-to-white scale.

`trimPNG` If `TRUE` (the default), the PhyloPic PNG is trimmed to remove extraneous whitespace from the top and bottom, before rescaling of the color values of the PNG.

`colorGradient` Controls the depth gradient of color for the PhyloPics. For typical plotting in black color, this means adjusting the grayscale (and possibly removing any gray scale). Most of the silhouettes are binary black-and-white already but some aren't, but those gray-scale values (sometimes?) seem to exist to indicate very fine features. However, maybe an image is far too much gray-scale, in which case users can apply this argument. If `colorGradient = "original"` (the default), then nothing is adjusted. If `colorGradient = "trueMonochrome"`, the entire image's gradients are simplified to a duality: either fully colored or fully transparent. If `colorGradient = "increaseDisparity"`, then a slightly less extreme option is applied, with values transformed to greatly remove in-between gray-scale value, shifting them toward color or not-color without making the silhouette purely monochrome.

`...` Additional arguments, passed to `plot.phylo` for plotting of the tree. These additional arguments may be passed to `plot`, and from there to `plot`. Some arguments are reserved and cannot be passed, particularly: `direction`, `show.tip.label`, `no.margin`, `plot`, `xlim`, and`ylim`.

Details

This function preferably will pull the identifiers for which images are to be associated with the tip taxa from `taxaDataPBDB$image_no`. By default, `taxaDataPBDB` itself is assumed to be an element of `tree` named `tree$taxaData`, as the PBDB data table used to construct the tree is appended to the output tree when `makePBDBtaxonTree` is used to construct a taxonomy-tree. If the functions listed in `getDataPBDB` are used to obtain the taxonomic data, this table will include the `image_no` variable, which is the image identifier numbers needed to call PNGs from the Paleobiology Database API. If `taxaDataPBDB` isn't provided, either by the user directly, or as an element of `tree`.

Value

This function silently returns the positions for elements in the tree (.e. the environmental information obtained about the previous plotting environment of the tree as plotted), along with a saved set of the graphic parameters as they were at the end of the function's run.

Author(s)

David W. Bapst

References

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

See Also

See getDataPBDB, makePBDBtaxonTree, and plotPhyloPicTree.

Examples

```
library(paleotree)

taxaAnimals<-c("Archaeopteryx", "Eldredgeops",
"Corvus", "Acropora", "Velociraptor", "Gorilla",
"Olenellus", "Lingula", "Dunkleosteus",
"Tyrannosaurus", "Triceratops", "Giraffa",
"Megatheriidae", "Aedes", "Histiodella",
"Rhynchotrema", "Pecten", "Homo", "Dimetrodon",
"Nemagraptus", "Panthera", "Anomalocaris")

data <-getSpecificTaxaPBDB(taxaAnimals)
tree <- makePBDBtaxonTree(data, rankTaxon = "genus")

plotPhyloPicTree(tree = tree)

# let's plot upwards but at a funny size
dev.new(height = 5, width = 10)
plotPhyloPicTree(tree = tree,
 orientation = "upwards")

# dated tree plotting

#date the tree
timeTree <- dateTaxonTreePBDB(tree, minBranchLen = 10)

plotPhyloPicTree(tree = timeTree)

# plotting the dated tree with an axis
plotPhyloPicTree(tree = timeTree,
depthAxisPhylo= TRUE)

# now upwards!
plotPhyloPicTree(tree = timeTree,
orientation = "upwards",
depthAxisPhylo= TRUE)
```

```
###################################
# plotting a time tree with stratigraphic ranges

plotPhyloPicTree(tree = timeTree,
   addTaxonStratDurations = TRUE)

plotPhyloPicTree(tree = timeTree,
   addTaxonStratDurations = TRUE,
   orientation = "upwards",
   depthAxisPhylo= TRUE)

########
# adjusting a tree to ignore a very old root

# let's pretend that metazoans are extremely old
treeOldRoot <- timeTree
rootEdges <- timeTree$edge[,1] == (Ntip(timeTree)+1)
rootEdgeLen <- timeTree$edge.length[rootEdges]
treeOldRoot$edge.length[rootEdges] <- rootEdgeLen + 1500
treeOldRoot$root.time <- NULL

# plot it
plot(treeOldRoot)
axisPhylo()
# yep, that's really old

# let's plot it now with the PhyloPic
plotPhyloPicTree(tree = treeOldRoot,
depthAxisPhylo = TRUE)

# let's crop that old lineage
plotPhyloPicTree(tree = treeOldRoot,
maxAgeDepth = 500,
depthAxisPhylo = TRUE)
# cool!

###################################
# playing with colors
plotPhyloPicTree(tree = tree,
 taxaColor = "green")

# inverting the colors
par(bg="black")
taxaColors <- rep("white",Ntip(tree))
# making a red giraffe
taxaColors[4] <- "red"
plotPhyloPicTree(
tree = tree,
orientation = "upwards",
edge.color = "white",
taxaColor=taxaColors)
```

```
#####################################
# let's try some different phylopics
  # like a nice tree of commonly known tetrapods

tetrapodList<-c("Archaeopteryx", "Columba", "Ectopistes",
"Corvus", "Velociraptor", "Baryonyx", "Bufo",
"Rhamphorhynchus", "Quetzalcoatlus", "Natator",
"Tyrannosaurus", "Triceratops", "Gavialis",
"Brachiosaurus", "Pteranodon", "Crocodylus",
"Alligator", "Giraffa", "Felis", "Ambystoma",
  "Homo", "Dimetrodon", "Coleonyx", "Equus",
"Sphenodon", "Amblyrhynchus")

data <-getSpecificTaxaPBDB(tetrapodList)

tree <- makePBDBtaxonTree(data, rankTaxon = "genus")

plotPhyloPicTree(tree = tree)



#####################################
## Not run:
# let's check out speed increase from caching!
    # can try this on your own machine

#first time
system.time(plotPhyloPicTree(tree = tree))
# second time
system.time(plotPhyloPicTree(tree = tree))

#################################
# make a pretty plot

taxaSeventyEight <- c(
"Archaeopteryx", "Pinus", "Procoptodon", "Olenellus", "Eldredgeops",
"Quetzalcoatlus", "Homo", "Tyrannosaurus", "Triceratops", "Giraffa",
"Bolivina", "Cancer", "Dicellograptus", "Dunkleosteus", "Solanum",
"Anomalocaris", "Climacograptus", "Halysites", "Cyrtograptus",
"Procoptodon", "Megacerops", "Moropus", "Dimetrodon", "Lingula",
"Rhynchosaurus", "Equus", "Megaloceros", "Rhynchotrema", "Pecten",
"Echinaster", "Eocooksonia", "Neospirifer", # "Prototaxites",
"Cincinnaticrinus", "Nemagraptus", "Monograptus", "Pongo", "Acropora",
"Histiodella", "Agathiceras", "Juramaia", "Opabinia", "Arandaspis",
"Corvus", "Plethodon", "Latimeria", "Phrynosoma", "Araucarioxylon",
"Velociraptor", "Hylonomus", "Elginerpeton", "Rhyniognatha",
"Tyto", "Dromaius", "Solenopsis", "Gorilla", "Ginkgo", "Terebratella",
"Caretta", "Crocodylus", "Rosa", "Prunus", "Lycopodium", "Meganeura",
"Diplodocus", "Brachiosaurus", "Hepaticae", "Canadaspis", "Pikaia",
"Smilodon", "Mammuthus", "Exaeretodon", "Redondasaurus", "Dimetrodon",
"Megatheriidae", "Metasequoia", "Aedes", "Panthera", "Megalonyx")

data <-getSpecificTaxaPBDB(taxaSeventyEight)
```

```
tree <- makePBDBtaxonTree(data, rankTaxon = "genus")

timeTree <- dateTaxonTreePBDB(tree,
  minBranchLen = 10)

date <- format(Sys.time(), "%m-%d-%y")
file <- paste0(
"tree_taxa78_phylopic_stratTree_",
date, ".pdf")

png(file = file,
height = 5, width = 12,
units = "in", res = 300)
par(bg="black")
par(mar=c(0,0,3,0))
taxaColors <- rep("white", Ntip(timeTree))
taxaColors[4] <- "red"

plotPhyloPicTree(
tree = timeTree,
orientation = "upwards",
addTaxonStratDurations = TRUE,
edge.color = "white",
maxAgeDepth = 700,
taxaColor=taxaColors,
depthAxisPhylo = TRUE,
colorAxisPhylo = "white")
dev.off()
shell.exec(file)


## End(Not run)
```

---

plotTraitgram *Plot a Traitgram for Continuous Traits*

Description

plotTraitgram plots a traitgram showing the evolution of a continuous trait. If node values are not given (i.e. the data is empirical data collected from tips, rather than simulated data), maximum-likelihood ancestral trait estimation is used to calculate node values. (Ackerly, 2009) given a tree and a set of continuous trait values.

Usage

```
plotTraitgram(trait, tree, main = "", conf.int = TRUE, lwd = 1.5)
```

Arguments

trait	A vector of continuous trait values. If the length of trait is equal to the number of tips and number of nodes, than it is presumed internal node values are given. If the length of trait is equal to the number of tips of tree then these will be treated as tip values and ancestral trait reconstruction will be used to reconstruct the missing ancestral values. If trait is not named, or if internal node values are given, then values will be presumed to be in the order of tip/node numbering in tree$edge, which for tips is the same as the ordering in tree$tip.label.
tree	A phylo object.
main	Main title of traitgram plot.
conf.int	If TRUE (the default), confidence intervals are plotted.
lwd	The line width used for branches in the figure.

Details

By default, this function will use ace from the library ape to reconstruct ancestral traits and confidence intervals using the PIC method, if internal node values (i.e. ancestral node values) are not given.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

Return no value, just plot the traitgram.

Note

One should probably never do ancestral trait estimation without looking at the confidence intervals, as these reconstructed estimates tend to be very uncertain.

Author(s)

David W. Bapst

References

Ackerly, D. 2009 Conservatism and diversification of plant functional traits: Evolutionary rates versus phylogenetic signal. *Proceedings of the National Academy of Sciences* **106**(Supplement 2):19699–19706.

See Also

ace

Also see the functions traitgram in the library picante and phenogram in the library phytools.

Examples

```
set.seed(444)
tree <- rtree(10)
trait <- rTraitCont(tree)

#first, traitgram without conf intervals
plotTraitgram(trait,tree,conf.int = FALSE)

#now, with
plotTraitgram(trait,tree)
#not much confidence, eh?

# plotting simulated data
    # with values for ancestral nodes as input
trait <- rTraitCont(tree, ancestor = TRUE)
plotTraitgram(tree = tree,trait = trait)
```

---

pqr2Ps *Joint Probability of A Clade Surviving Infinitely or Being Sampled Once*

Description

Given the rates of branching, extinction and sampling, calculates the joint probability of a random clade (of unknown size, from 1 to infinite) either (a) never going extinct on an infinite time-scale or (b) being sampled at least once, if it does ever go extinct. As we often assume perfect or close to perfect sampling at the modern (and thus we can blanket assume that living groups are sampled), we refer to this value as the Probability of Being Sampled, or simply P(s). This quantity is useful for calculating the probability distributions of waiting times that depend on a clade being sampled (or not).

Usage

```
pqr2Ps(p, q, r, useExact = TRUE)
```

Arguments

p	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.
q	Instantaneous rate of extinction (mu)
r	Instantaneous rate of sampling (per taxon, per time-unit).
useExact	If TRUE, an exact solution developed by Emily King is used; if FALSE, an iterative, inexact solution is used, which is somewhat slower (in addition to being inexact...).

Details

Note that the use of the word 'clade' here can mean a monophyletic group of any size, including a single 'species' (i.e. a single phylogenetic branch) that goes extinct before producing any descendants. Many scientists I have met reserve the word 'clade' for only groups that contain at least one branching event, and thus contain two 'species'. I personally prefer to use the generic term 'lineage' to refer to monophyletic groups of one to infinity members, but others reserve this term for a set of morphospecies that reflect an unbroken anagenetic chain.

Obviously the equation used makes assumptions about prior knowledge of the time-scales associated with clades being extant or not: if we're talking about clades that originated a short time before the recent, the clades that will go extinct on an infinite time-scale probably haven't had enough time to actually go extinct. On reasonably long time-scales, however, this infinite assumption should be reasonable approximation, as clades that survive 'forever' in a homogenous birth-death scenario are those that get very large immediately (similarly, most clades that go extinct also go extinct very shortly after originating... yes, life is tough).

Both an exact and inexact (iterative) solution is offered; the exact solution was derived in an entirely different fashion but seems to faithfully reproduce the results of the inexact solution and is much faster. Thus, the exact solution is the default. As it would be very simple for any user to look this up in the code anyway, here's the unpublished equation for the exact solution:

$Ps = 1 - (((p + q + r) - (sqrt(((p + q + r)^2) - (4 * p * q))))/(2 * p))$

The above exact solution was independent derived and published by Didier et al. (2017). Also see Wagner (2019) for additional discussion of this value and its importance for understanding the timing of branching events in an imperfect fossil record.

Value

Returns a single numerical value, representing the joint probability of a clade generated under these rates either never going extinct or being sampled before it goes extinct.

Author(s)

This function is entirely the product of a joint unpublished effort between the package author (David W. Bapst), Emily A. King and Matthew W. Pennell. In particular, Emily King solved a nasty bit of calculus to get the inexact solution and later re-derived the function with a quadratic methodology to get the exact solution. Some elements of the underlying random walk model were provided by S. Nalayanan (a user on the website stackexchange.com) who assisted with a handy bit of math involving Catalan numbers.

References

Bapst, D. W., E. A. King and M. W. Pennell. Unpublished. Probability models for branch lengths of paleontological phylogenies.

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Didier, G., M. Fau, and M. Laurin. 2017. Likelihood of Tree Topologies with Fossils and Diversification Rate Estimation. *Systematic Biology* 66(6):964-987.

Wagner, P. J. 2019. On the probabilities of branch durations and stratigraphic gaps in phylogenies of fossil taxa when rates of diversification and sampling vary over time. *Paleobiology* 45(1):30-55.

See Also

SamplingConv

Examples

```
#with exact solution
pqr2Ps(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    useExact = TRUE
    )

#with inexact solution
pqr2Ps(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    useExact = TRUE
    )
```

---

probAnc *Probability of being a sampled ancestor of another sampled taxon*

Description

This function uses models from Foote (1996) to calculate the probability of sampling a descendant of a morphotaxon in the fossil record, given the sampling probability and estimates of origination and extinction rates.

Usage

```
probAnc(
  p,
  q,
  R,
  mode = "budding",
  analysis = "directDesc",
  Mmax = 85,
  nrep = 10000
)
```

Arguments

p	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.

q	Instantaneous rate of extinction (mu)
R	Per-interval probability of sampling a taxon at least once.
mode	Mode of morphotaxon differentiation, based on definitions in Foote, 1996. Can be pure cladogenetic budding ("budding"), pure cladogenetic bifurcating ("bifurcating") or pure anagenetic within-lineage change ("anagenesis"; i.e. Foote's 'phyletic change'). Default mode is "budding".
analysis	The type of analysis to be performed, either the probability of sampling direct descendants ("directDesc") or of sampling indirect descendants ("indirectDesc").
Mmax	The maximum number of direct descendants (M) to sum over in the function, which is ideally meant to be a sum from zero to infinity, like nrep. Unfortunately, (2*M) is used in a factorial, which means we are limited to a relatively small upper bound on M.
nrep	Number of repetitions to run in functions which are meant to sum over infinity. Default is arbitrarily high.

Details

These values are always calculated assuming infinite time for the potential ancestor to produce daughter taxa (assuming it lives that long) and under homogenous birth, death and sampling rates/probabilities, which is a situation that may be overly ideal relative to many real fossil records.

These probabilities can be calculated for either direct descendants, i.e. the probability of sampling any morphotaxa that arise immediately from the particular morphotaxon that could be an ancestor, or indirect descendants, i.e. the probability for any morphotaxon that has the morphotaxon of question as an ancestor, no matter how distant. See the argument analysis for details. Mode of differentiation can also be varied for three different models, see the argument mode.

The probability of sampling a taxon's ancestor is calculated while accounting for the probability that extinction might occur before any descendants are produced. Thus, if p = q, the probability of a taxon going extinct before it produces any descendants will be 0.5, which means that even when sampling is perfect (R = 1, meaning completeness of 100 can be no higher than 0.5. See Foote (1996) for a graphic depiction of this non-intuitive ceiling. For reasons (probably?) having to do with finite approximations of infinite summations, values close to perfect sampling may have values slightly higher than this ceiling, which is also apparent visually in the figures in Foote (1996). Thus, values higher than 0.5 when p = q should be discounted, and in general when sampling rate is high, results should be treated cautiously as overestimates.

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

See Also

SamplingConv

Examples

```
# examples, run at very low nrep for sake of speed (examples need to be fast)
```

```
# default options
    # probability of sampling a direct descendant
probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "budding",
        analysis = "directDesc",
        nrep = 100)

# other modes
probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "bifurcating",
        analysis = "directDesc",
        nrep = 100)
probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "anagenesis",
        analysis = "directDesc",
        nrep = 100)

# probability of having sampled indirect descendants of a taxon

# first, the default
probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "budding",
        analysis = "indirectDesc",
        nrep = 100)

probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "bifurcating",
        analysis = "indirectDesc",
        nrep = 100)

probAnc(p = 0.1, q = 0.1, R = 0.5,
        mode = "anagenesis",
        analysis = "indirectDesc",
        nrep = 100)
```

`RaiaCopesRule` *Dated Trees and Trait Data for Ammonites, Ceratopsians and Cervids from Raia et al. 2015*

Description

Dated phylogenetic trees for fossil ammonite genera, fossil ceratopsian species and (both extinct and extant) cervid species, as well as trait data (shell diameter and fractal complexity of the first suture) for the ammonite dataset, taken from the recent publication by Raia et al. (2015) in The American Naturalist. The goal of this paper was to examine the relationship between ornamental complexity and body size in three very different groups, but the datasets are very relatively large and useful for demonstrating application of comparative methods to fossil trees.

Format

The attached datasets consist of three phylogenetic trees as `phylo` objects, a `data.frame` consisting of three traits for ammonites (the third trait is the log of stratigraphic duration), and the two physical traits (shell size and suture complexity) as separate vectors, with taxon names.

Details

It appears that the trees were dated with tips at the last appearances, although this doesn't appear to be explicitly stated in Raia et al.

Source

These datasets were taken from the following study: Raia, P., F. Passaro, F. Carotenuto, L. Maiorino, P. Piras, L. Teresi, S. Meiri, Y. Itescu, M. Novosolov, M. A. Baiano, R. Martinez, and M. Fortelius. 2015. Cope's Rule and the Universal Scaling Law of Ornament Complexity. *The American Naturalist*. 186(2):165-175.

And the corresponding Dryad repository: Raia P, Passaro F, Carotenuto F, Maiorino L, Piras P, Teresi L, Meiri S, Itescu Y, Novosolov M, Baiano MA, Martinez R, Fortelius M (2015) Data from: Cope's rule and the universal scaling law of ornament complexity. *Dryad Digital Repository*. (https://doi.org/10.5061/dryad.50dr8)

See Also

retiolitinae, macroperforateForam

Examples

```
data(RaiaCopesRule)

# plotting trees
plot(ladderize(ammoniteTreeRaia));axisPhylo()

plot(ceratopsianTreeRaia);axisPhylo()

plot(cervidTreeRaia);axisPhylo()

# plotting traitgrams for ammonites
plotTraitgram(tree = multi2di(ammoniteTreeRaia), trait = sutureComplexity,
 conf.int = FALSE, main = "Ammonite Suture Complexity")

plotTraitgram(tree = multi2di(ammoniteTreeRaia), trait = shellSize,
 conf.int = FALSE, main = "Ammonite Shell Diameter")

##############################################################################################

# The data set was generated by sourcing the following script:



library(paleotree)
```

```
# Let's read in the trees from Raia et al 2015, AmNat
# following is taken from their supplemental appendix, available at AmNat
# they all appear to be trees dated to the last appearance times
# *and* specifically the end-boundary of the interval containing the last appearance

########################################
# ammonite genera

ammoniteTreeRaia <- paste0("(((((Araxoceras:4,Eoaraxoceras:4)Araxoceratidae:26.5,Pseudasp",
"idites:33.199997,Dieneroceras:37.300003,(Tardicolumbites:13.000015,Cowboyiceras:13.000023)",
"Dinaritaceae:24.299988,Grambergia:42.5,(Amphipopanoceras:6, Megaphyllites:46.399994)Megaph",
"yllitaceae:36.5,(Proteusites:11,Nathorstites:21)Nathorstitaceae:31.5,(Inyoites:7,Lanceolit",
"es:7,Parussuria:7)Noritaceae:30.300003,(((Placites:66.700012,((Acrochordiceras:10.199997, ",
"Bradyia:10.199997,Globacrochordiceras:5,Paracrochordiceras:10.199997)Acrochordiceratidae:9",
".000015,Balatonites:19.200012,(Favreticeras:10,Guexites:10,Gymnotoceras:10)Beyrichitidae:9",
".200012, Eogymnotoceras:19.200012,Goricanites:14.000015)Ceratitaceae:7.100006)clade_16:7.0",
"99976, (((Gaudemerites:13.000015,(Owenites:9.000015,Prosphingites:9.000015)Paranannitidae:",
"4,Meekoceras:13.000015, Arctoceras:13.000015)Meekoceratoidea:5.06665,(((Riedelites:85.6000",
"06,((((Berriasella:15.399994, (Polyptychites:20.399994,Surites:14)Polyptychitidae:1.399994",
")clade_32:1.400002,Bodrakiceras:20.300003, Busnardoites:16.800003,Campylotoxia:20.300003,K",
"arakaschiceras:23.199997,Luppovella:16.800003,Malbosiceras:13, Pomeliceras:13.399994)Neoco",
"mitidae:21.199997,(Otohoplites:8.199997,Sonneratia:4.5,Anadesmoceras:4.5, Anahoplites:20,A",
"rcthoplites:8.199997,Cleoniceras:8.199997,Dimorphoplites:8.199997,Epihoplites:20, Gastropl",
"ites:13.900002,Grycia:13.900002,Hoplites:13.900002)Hoplitidae:60.899994)clade_29:20.200005",
", (Engonoceras:20.400002,(Knemiceras:16.400002,Parengonoceras:7,Platiknemiceras:7)Knemicer",
"atidae:4) Engonoceratoidea:74.600006,(((Glochiceras:11,(Aconeceras:36.799995,Falciferella:3",
"5.899994,Protaconeceras:7, Sanmartinoceras:24.369995)Oppeliidae:25.400009)Haplocerataceae:",
"15.775009,(((Mortoniceras:16.300003, Oxytropidoceras:14)Brancoceratidae:27.633331,((Parado",
"lphia:12.700005,Stoliczkaia:18.800003,Tegoceras:7) Lyelliceratidae:7.566666,((Borissiakoce",
"ras:10.5,Mammites:7,Mantelliceras:12.800003)Acanthoceratidae:11.783333, (Neoptychites:6,Va",
"scoceras:6)Vascoceratidae:12.783333)clade_49:11.783333)clade_47:7.566666)clade_45:7.566666",
", (Epileymeriella:5,Leymeriella:11.099998)Leymeriellidae:30.400002,(Beudanticeras:44.03332",
"5, Burckhardtites:21.303329,(Barremites:1.666672,Desmoceras:48.166672)clade_55:1.666656, P",
"seudohaploceras:21.303329,Pseudosaynella:21.303329,Pseudosilesites:21.303329,(Puzosia:56.6",
"50002, (Forbesiceras:27.666664,(Melchiorites:6.083328,Uhligella:15.48333)clade_58:6.083336",
")clade_57:6.083336) clade_56:6.083328,Valdedorsella:33.633331,Zuercherella:33.73333)Desmoc",
"eratidae:1.666667) Acanthocerataceae:42.575012)clade_39:15.774994,((Coroniceras:1.25,(Mega",
"tyloceras:76.203336, (Zugodactylites:10.016663,Amaltheus:2.616669)Eoderocerataceae:2.61666",
"9)clade_61:2.616669)clade_60:1.25, Oxynoticeras:9.100006)Psiloceratacea:1.25)clade_38:1.2",
"5)Ammonitina:4,((Saghalinites:14,Tetragonites:14) Tetragonitidae:22,(Eogaudryceras:4,Gaudr",
"yceras:32,Zelandites:32)Gaudryceratidae:4)Tetragonitoidea:97.100006, (Costidiscus:12.00000",
"8,Macroscaphites:34.860008)Macroscaphitidae:64.139999)Ammonitida:30.222214, (Ammonitoceras",
":98.570007,Argonauticeras:98.570007,Audaxlytoceras:27.600006,Holcolytoceras:21, (Eulytocer",
"as:65.713333,Jaubertella:78.043335)clade_84:32.85667,(Ectocentrites:9.433334,(Adnethiceras",
":8.166656, Galaticeras:14.766663)clade_87:8.166672,((Protetragonites:56.933334,Lytoceras:5",
"0.833336)clade_89:50.833344, Pleuroacanthites:4.666672)clade_88:4.666656)Pleuroacanthitida",
"e:4.666667,Pterolytoceras:65.100006) Psiloceratida:18.222214)clade_26:18.222229,((Juraphyl",
"lites:6,Nevadaphyllites:6,Togaticeras:6, Tragophylloceras:12.600006)Juraphyllitidae:6,Hypo",
"rbulites:107.300003,(Adabofoloceras:25.400009, Hypophylloceras:121.100006,Ptychophyllocera",
"s:56.600006,Salfeldiella:56.600006,Holcophylloceras:61.150009, Phylloceras:121.100006,Leio",
"phylloceras:46.800003)Phylloceratidae:15)Phylloceratida:45.444443)clade_25:18.222214) clad",
"e_22:5.066681,(Paranannites:11.566666,(Proarcestes:8.383331,Ptychites:8.383331)clade_94:8.",
```

```
"383331) clade_93:11.566681)clade_21:5.06665)clade_15:7.100006,(Deweveria:33.300003,Juvenit",
"es:33.300003,(Cibolites:11.5, Kingoceras:22.5,Meitianoceras:24.199997,Paraceltites:4)Parac",
"eltitidae:4,Preflorianites:33.300003, Xenodiscus:33.300003)Xenodiscoidea:2)clade_14:2,Cart",
"eria:37.300003,Courtilloticeras:37.300003, Eschericeratites:37.300003,Tapponnierites:37.30",
"0003)Ceratitida:101.025024,(((Daraelites:76.399994, Epicanites:23.299988,Praedaraelites:15",
")Daraelitidae:28.300018,(Becanites:19.900024,Dombarocanites:47.100006, Eocanites:19.900024",
",Merocanites:22.400024,Prolecanites:4.5)Prolecanitidae:4.5)Prolecanitina:1,((Neopronorites",
":7, Sakmarites:14)Pronoritidae:17.5,(Artinskia:4.5,Bamyaniceras:34.5,Medlicottia:40.5,Prop",
"inacoceras:31.5,Synartinskia:20, Uddenites:4.5)Medlicottiidae:4.5)Medlicottiina:66.700012)",
"Prolecanitida:14.825012)clade_3:14.824982, (((Raymondiceras:6,(Dimeroceras:5,Paratornocera",
"s:5)Dimeroceratidae:5)Dimeroceratoceae:10,((Acrocanites:7, Jdaidites:7)Acrocanitidae:16.40",
"0024,Kazakhstania:25.900024,Praeglyphiloceras:8,(Imitoceras:10.900024,Prionoceras:4, Triim",
"itoceras:19.400024)Prionoceratidae:4,(Maeneceras:1,Sporadoceras:4)Sporadoceratidae:4)Prion",
"oceratoceae:12, (Pseudoclymenia:5,(Discoclymenia:4.5,Posttornoceras:4.5)Posttornoceratidae",
":4.5)Tornoceratoceae:11)Tornoceratina:10, (Popanoceras:117.533356,((Epitornoceras:28,Falci",
"tornoceras:28,Lobotornoceras:6.300018,Protornoceras:5, Tornoceras:18.100006)Tornoceratidae",
":0.666656,(Cheiloceras:13,Torleyoceras:13)Cheiloceratidae:15.666656, Polonoceras:28.666656",
")Cheiloceratoceae:0.666687,((((Kargalites:30.344452,(Adrianites:22.5,Nevadoceras:11, Veruz",
"hites:11)Adrianitidae:19.344452)clade_122:19.344452,Pintoceras:25.78891)clade_121:25.78887",
"9, ((Waagenoceras:53.14447,((Metalegoceras:9,Pericycloceras:12)Metalegoceratidae:9.5,Uralo",
"ceras:14) Neoicoceratoidea:25.64447)clade_125:25.64444,((Brannerоceras:3,Diaboloceras:10.6",
"49994,Paralegoceras:63.600006, Schistoceras:38.100006)Schistoceratidae:3,(Wellerites:10.64",
"9994,Winslowoceras:3)Welleritidae:3) Schistoceratoceae:17.188904)clade_124:17.188873)clade",
"_120:17.188904,(Antegoniatites:9,Habadraites:9, Primogoniatites:9,Progoniatites:9)Goniatit",
"idae:17.866699)clade_119:17.866669,(Dzhaprakoceras:23,(Follotites:18.5, Muensteroceras:10,",
"Xinjiangites:10)Muensteroceratidae:2,(Ammonellipsites:12.5,Helicocyclus:10,Nodopericyclus:",
"10, Ouaoufilalites:10,Pericyclus:10)Pericyclidae:10.5,(Eurites:10.25,Mouydiria:10.25,Rotop",
"ericyclus:10.25) Rotopericlydae:10.25,(Jerania:6,Kusinia:6,Temertassetia:6)Temertassetiida",
"e:14.5)Pericyclaceae:24.233368, Stacheoceras:136.033356)Goniatitina:0.666667)Goniatitida:9",
".649994)clade_2:9.649994,(Gyroceratites:14.799988, ((Teicherticeras:8.93335,((((Probelocer",
"as:36,(Timanites:12.600006,(Darkaoceras:2.5,Keuppites:2.5)Taouzitidae:2.5, (Gogoceras:10.1",
"00006,Pseudoprobeloceras:2.5)Ponticeratidae:2.5,(Beloceras:9,Mesobeloceras:9) Beloceratida",
"e:3.600006,(Archoceras:23.399994,Manticoceras:8,Mixomanticoceras:8,Sphaeromanticoceras:8) ",
"Gephuroceratidae:4.600006)Gephuroceratoceae:8)Gephuroceratatina:1.033325,Agoniatites:14.03",
"3325) clade_144:1.033325,Celaeceras:6.866669)clade_143:1.033325,((Werneroceras:0.399994,(S",
"obolewia:0.200012, (((Cyrtoclymenia:2.5,Clymenia:2.5)Clymeniina:2.5,Protoxyclymenia:5,Plat",
"yclymenia:5)Clymeniida:21.066681, (Lunupharciceras:1.533325,Pharciceras:9.133331,Stenophar",
"ciceras:1.533325,Synpharciceras:1.533325) Pharciceratatina:1.533356)clade_154:1.533325)cla",
"de_153:0.199982)clade_152:0.200002,Anarcestes:5) Anarcestina:11.099976)clade_142:1.033356)",
"clade_141:1.033325,Anetoceras:9.966675)clade_140:1.03333) Agoniatitida:7.100006)clade_1;")

ammoniteTreeRaia <- read.tree(text = ammoniteTreeRaia)

# what about the root age?
# Raia et al. are unclear
 # however... ahandful of taxa are known to last occur at the end-Cretaceous mass ext
 # Phylloceras
#
# Latest occurring tips are:
ammoniteTreeRaia$tip.label[
 which(node.depth.edgelength(ammoniteTreeRaia) == max(node.depth.edgelength(ammoniteTreeRaia)))]
#
# so we can treat distance of Phylloceras from root + end Cretaceous (66.043 Ma) as $root.time
```

```
(ammoniteTreeRaia$root.time <- 66.043+
 node.depth.edgelength(ammoniteTreeRaia)[which(ammoniteTreeRaia$tip.label == "Phylloceras")])

# now let's plot it
plot(ladderize(ammoniteTreeRaia));axisPhylo()


## Not run:

# and let's load trait data from Raia et al. Appendix B:
 # FD = fractal dimension of first suture (suture complexity)
 # Log D = log of the mean shell diameter per genus (body size)
 # log dur = log of the stratigraphic duration in million years.
ammoniteTraitsRaia <- read.table("ammoniteTraitsRaia.txt",row.names = 1,header = TRUE)

sutureComplexity <- ammoniteTraitsRaia$FD
shellSize <- ammoniteTraitsRaia$Log_D
names(shellSize) <- names(sutureComplexity) <- rownames(ammoniteTraitsRaia)

plotTraitgram(tree = multi2di(ammoniteTreeRaia), trait = sutureComplexity,
  conf.int = FALSE, main = "Ammonite Suture Complexity")

plotTraitgram(tree = multi2di(ammoniteTreeRaia), trait = shellSize,
  conf.int = FALSE, main = "Ammonite Shell Diameter")


## End(Not run)

########################################
# ceratopsian species

ceratopsianTreeRaia <- paste0("((((((((((((Centrosaurus_apertus:5.1,Styracosaurus_alberte",
 "nsis:5.9):1,(((Pachyrhinosaurus_perotorum:10.5,Pachyrhinosaurus_lakustai:7):0.5,Achelousau",
 "rus_horneri:6.3):0.5,Einiosaurus_procurvicornis:6.5):1):0.5, Avaceratops_lammersi:5.5):0.5",
 ",Diabloceratops_eatoni:3):1.1,((Chasmosaurus_russelli:1.4,Chasmosaurus_belli:1.6):2.5, (Mo",
 "joceratops_perifania:3.7,(Agujaceratops_mariscalensis:1.9,((Pentaceratops_sternbergii:3.5,",
 " Utahceratops_gettyi:1):1.5,((Vagaceratops_irvinensis:1.3,Kosmoceratops_richardsoni:1):0.4",
 ",(Anchiceratops_ornatus:3.9, (Arrhinoceratops_brachyops:3.9,(Torosaurus_latus:3,(Tricerato",
 "ps_horridus:2, Triceratops_prorsus:2):1):6):0.5):1.7):1):0.5):0.5):0.5):3.8):12.9,(Bagacer",
 "atops_rozhdestvenskyi:17, (Protoceratops_hellenikorhinus:9.5,Protoceratops_andrewsi:9.5):1",
 "2):4.5):6,(Prenoceratops_pieganensis:21, Leptoceratops_gracilis:31.6):4.5):7.5,Archaeocera",
 "tops_oshimai:6):5,Auroraceratops_rugosus:15):21, Liaoceratops_yanzigouensis:6):4,(Hongshan",
 "osaurus_houi:9,(Psittacosaurus_mongoliensis:33.5, (Psittacosaurus_meileyingensis:20,(Psitt",
 "acosaurus_major:7.5,(Psittacosaurus_gobiensis:21,(Psittacosaurus_sinensis:24, Psittacosaur",
 "us_neimongoliensis:18):1):1.5):0.5):0.5):0.5):1):23,Yinlong_downsi:6):3;")

ceratopsianTreeRaia <- read.tree(text = ceratopsianTreeRaia)

# Raia et al. placed origin of ceratopsians at ~163 Ma, base of Oxfordian
ceratopsianTreeRaia$root.time <- 163

plot(ceratopsianTreeRaia);axisPhylo()
```

```
###############################################
# cervid species

cervidTreeRaia <- paste0("((((Lagomeryx_parvulus:9.925998,Lagomeryx_pumilio:10.775998):3.",
 "25,(Procervulus_flerovi:11.425998,Procervulus_dichotomus:7.025998,Procervulus_praelucidus:",
 "5.675998):3.25,(Stephanocemas_aralensis:6.925998, Stephanocemas_thomsoni:11.175998):2):2,(",
 "((Euprox_furcatus:14.440997,Euprox_minimus:12.590997, Euprox_dicranoceros:14.190997):2.185",
 "001,Heteroprox_larteti:12.175998):1.5,Muntiacus_muntjak:25.531498):1.5):1.5, (((((Alces_la",
 "tifrons:7.151589758,Alces_alces:7.245998):2.29,Cervalces_scotti:9.525768):6.64, Rangifer_t",
 "arandus:16.175998):4.35,(Procapreolus_loczyi:17.840998,Capreolus_capreolus:17.905998):2.62",
 "5):5.25, (((Cervavitus_novorossiae:9.109332,Cervavitus_variabilis:9.379332):7.149999, Plio",
 "cervus_pentelici:13.069331):2.966667,(((((Dama_clactoniana:5.133775345,Dama_dama:5.199332)",
 ":2.903333, (Pseudodama_farnetensis:5.860846548,Pseudodama_lyra:4.242887928, Pseudodama_nes",
 "tii:5.762011259):2.083333):4.166667,(Eucladoceros_ctenoides:6.892665, Eucladoceros_dicrani",
 "os:7.692563015):4.166667):2.083333,((Cervus_elaphus:5.734332,Cervus_nippon:5.744332, Rusa_",
 "timorensis:5.740332,Rusa_unicolor:5.744332,Cervus_duvaucelii:5.671332):3.4, Croizetoceros_",
 "ramosus:7.834332):5.208333):2.083333,((Praemegaceros_verticornis:9.610727017, (Megaceroide",
 "s_obscurus:6.084504245,Megaceroides_solilhacus:6.725161676):2.883334):2.883333, (Megalocer",
"os_savini:7.349060017,Megaloceros_giganteus:7.430999):5.145):3.849999):6.6):2.75):2.75);")

cervidTreeRaia <- read.tree(text = cervidTreeRaia)

# Many of the latest-occurring tips are still extant, like Rusa unicolor and Dama dama:
cervidTreeRaia$tip.label[
which(node.depth.edgelength(cervidTreeRaia) == max(node.depth.edgelength(cervidTreeRaia)))]

# note!
# if you plot the tree there seem to be a lot more taxa that are *almost* as late-occurring
# unclear if this is recently extinct taxa, computational rounding error, or what

# so we can treat distance of Dama dama to root as $root.time
(cervidTreeRaia$root.time <-
node.depth.edgelength(cervidTreeRaia)[which(cervidTreeRaia$tip.label == "Dama_dama")])

plot(cervidTreeRaia);axisPhylo()


## Not run:

save.image("RaiaCopesRule.rdata")


## End(Not run)
```

---

resolveTreeChar *Resolve Polytomies Using Parsimony-Based Reconstruction of a Discrete Character*

Description

This function resolves a set of given topology with less than fully-binary phylogenetic resolution so that lineages are shifted and internal nodes added that minimize the number of independent character transitions needed to explain an observed distribution of discrete character states for the taxa on such a tree, under various maximum-parsimony algorithms of ancestral character reconstruction, powered ultimately by function `ancestral.pars` in library `phangorn`. This function is mainly designed for use with poorly resolved trees which are being assessed with the function `minCharChange`.

Usage

```
resolveTreeChar(
  tree,
  trait,
  orderedChar = FALSE,
  stateBias = NULL,
  iterative = TRUE,
  cost = NULL,
  ambiguity = c(NA, "?"),
  dropAmbiguity = FALSE,
  polySymbol = "&",
  contrast = NULL
)
```

Arguments

`tree`	A cladogram of type `phylo`. Any branch lengths are ignored.
`trait`	A vector of trait values for a discrete character, preferably named with taxon names identical to the tip labels on the input tree.
`orderedChar`	Is the character of interest given for `trait` ordered or not? If FALSE (the default), then for each polytomy, all child nodes that appear to have the same state as the ancestor node will remain in the polytomy, and any additional states held by child nodes will each be grouped into their own unique polytomy that forms from a descendant node of the original polytomy. If TRUE, then the character will be reconstructed with a cost (step) matrix of a linear, ordered character, and polytomies will be resolved so that lineages with different states will be placed into a nested ladder that reflects the ordered character. As with the unordered option, child nodes with a state equivalent to the ancestral node will remain in the polytomy, while more primitive or more derived states will be sorted into their own separate ladders composed of paraphyletic groups, ordered so to move 'away' state-by-state from the ancestral node's inferred character state.
`stateBias`	This argument controls how `resolveTreeChar` handles ancestral node reconstructions that have multiple states competing for the maximum weight of any state (i.e. if states 0 and 1 both have 0.4 of the weight). The default, where `stateBias = NULL` causes uncertainty at nodes among states to be treated as a single 'group' identical to any states within it. Essentially, this means that for the example polytomy where the ancestor has maximum weight for both 0 and

1, any child nodes with 0, 1 or both of these states will be considered to have an identical state for the purpose of grouping nodes for the purpose of further resolving polytomies. If and only if `orderedChar = TRUE`, then additional options of `stateBias = 'primitive'` and `stateBias = 'derived'` become available, which instead force uncertain node assignments to either be the most primitive (i.e. the minimum) or the most derived (i.e. the maximum) among the maximum-weight states. In particular, `stateBias = 'primitive'` should favor gains and bias any analysis of character transitions against finding reversals.

`iterative` A logical argument which, if TRUE (the default), causes the function to repeat the polytomy-resolving functionality across the entire tree until the number of nodes stabilizes. If FALSE, polytomies are only passed a single time.

`cost` A matrix of the cost (i.e. number of steps) necessary to change between states of the input character trait. If `NULL` (the default), the character is assumed to be unordered with equal cost to change from any state to another. Cost matrices only impact the "MPR" algorithm; if a cost matrix is given but `type = "ACCTRAN"`, an error is issued.

`ambiguity` A vector of values which indicate ambiguous (i.e. missing or unknown) character state codings in supplied `trait` data. Taxa coded ambiguously as treated as being equally likely to be any state coding. By default, `NA` values and "?" symbols are treated as ambiguous character codings, in agreement with behavior of functions in packages `phangorn` and `Claddis`. This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package `phangorn`.

`dropAmbiguity` A logical. If `TRUE` (which is not the default), all taxa with ambiguous codings as defined by argument `ambiguity` will be dropped prior to ancestral nodes being inferred. This may result in too few taxa.

`polySymbol` A single symbol which separates alternative states for polymorphic codings; the default symbol is `"&"`, following the output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated by default with a string with state labels separated by an `"&"` symbol. For example, a taxon coded as polymorphic for states 1 or 2, would be indicated by the string `"1&2"`. `polySymbol` is used to break up these strings and automatically construct a fitting `contrast` table for use with this data, including for ambiguous character state codings.

`contrast` A matrix of type integer with cells of 0 and 1, where each row is labeled with a string value used for indicating character states in `trait`, and each column is labeled with the formal state label to be used for assign taxa to particular character states. A value of 1 indicates that the respective coding string for that row should be interpreted as reflecting the character state listed for that column. A coding could reflect multiple states (such as might occur when taxa are polymorphic for some morphological character), so the sums of rows and columns can sum to more than 1. If `contrast` is not `NULL` (the default), the arguments will nullify This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package phangorn. This structure is based on `phangorn`'s use of `contrasts` table used for statistical evaluation of factors. See the `phangorn` vignette "Special features of phangorn" for more details on its implementation within `phangorn` including an example. See examples below for the construction of an example contrast matrix for character data with

polymorphisms, coded as character data output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated with a string with state labels separated by an `"&"` symbol.

Details

As shown in the example code below, this function offers a wide variety of options for manipulating the maximum-parsimony algorithm used (i.e. MPR versus ACCTRAN), the ordering (or not) of character states, and potential biasing of uncertainty character state reconstructions (when ordered characters are assessed). This allows for a wide variety of possible resolutions for a given tree with polytomies and a discrete character. In general, the author expects that use of this function will be optimal when applied to ordered characters using one of the `stateBias` options, perhaps `stateBias = "primitive"` (based on theoretical expectations for slow evolving characters). However, anecdotal use of this function with various simulation datasets suggests that the results are quite variable, and so the best option needs to be assessed based on the prior assumptions regarding the data and the performance of the dataset with the various arguments of this function.

Value

Returns the resulting tree, which may be fully resolved, partly more resolved or not more resolved at all (i.e. have less polytomies) depending on what was possible, as constrained by ambiguities in character reconstructions. Applying `multi2di` is suggested as a post-step to obtain a fully-resolved cladogram, if one is desired.

Author(s)

David W. Bapst

References

Hanazawa, M., H. Narushima, and N. Minaka. 1995. Generating most parsimonious reconstructions on a tree: A generalization of the Farris-Swofford-Maddison method. Discrete Applied Mathematics 56(2-3):245-265.

Narushima, H., and M. Hanazawa. 1997. A more efficient algorithm for MPR problems in phylogeny. Discrete Applied Mathematics 80(2-3):231-238.

Schliep, K. P. 2011. phangorn: phylogenetic analysis in R. *Bioinformatics* 27(4):592-593.

Swofford, D. L., and W. P. Maddison. 1987. Reconstructing ancestral character states under Wagner parsimony. Mathematical Biosciences 87(2):199-229.

See Also

`ancPropStateMat` which is used internally by this function. This function was intentionally designed for use with `minCharChange`.

Examples

```
# let's write a quick&dirty ancestral trait plotting function

quickAncPlot <- function(tree, trait, cex, orderedChar = FALSE, type = "MPR", cost = NULL){
 ancData <- ancPropStateMat(tree = tree, trait = trait, orderedChar = orderedChar)
 ancCol <- (1:ncol(ancData))+1
  plot(tree,show.tip.label = FALSE,no.margin = TRUE,direction = "upwards")
  tiplabels(pch = 16,pie = ancData[(1:Ntip(tree)),],cex = cex,piecol = ancCol,
col = 0)
  nodelabels(pie = ancData[-(1:Ntip(tree)),],cex = cex,piecol = ancCol)
  }

##########

# examples with simulated data

set.seed(2)
tree <- rtree(50)
#simulate under a likelihood model
trait <- rTraitDisc(tree,k = 3,rate = 0.7)
tree <- degradeTree(tree,prop_collapse = 0.6)
tree <- ladderize(tree,right = FALSE)

#a bunch of type = MPR (default) examples
treeUnord <- resolveTreeChar(tree,trait,orderedChar = FALSE)
treeOrd <- resolveTreeChar(tree,trait,orderedChar = TRUE,stateBias = NULL)
treeOrdPrim <- resolveTreeChar(tree,trait,orderedChar = TRUE,stateBias = "primitive")
treeOrdDer <- resolveTreeChar(tree,trait,orderedChar = TRUE,stateBias = "derived")

#compare number of nodes
Nnode(tree) #original
Nnode(treeUnord) #unordered, biasStates = NULL, MPR
Nnode(treeOrd) #ordered, biasStates = NULL
Nnode(treeOrdPrim) #ordered, biasStates = 'primitive'
Nnode(treeOrdDer) #ordered, biasStates = 'derived'

#let's compare original tree with unordered-resolved tree
layout(1:2)
quickAncPlot(tree,trait,orderedChar = FALSE,cex = 0.3)
text(x = 43,y = 10,"Original",cex = 1.5)
quickAncPlot(treeUnord,trait,orderedChar = FALSE,cex = 0.3)
text(x = 43,y = 10,"orderedChar = FALSE",cex = 1.5)
#some resolution gained

#now let's compare the original and ordered, both biasStates = NULL
layout(1:2)
quickAncPlot(tree,trait,orderedChar = FALSE,cex = 0.3)
text(x = 43,y = 10,"Original",cex = 1.5)
quickAncPlot(treeOrd,trait,orderedChar = TRUE,cex = 0.3)
text(x = 43,y = 10,"orderedChar = TRUE",cex = 1.5)

#now let's compare the three ordered trees
layout(1:3)
quickAncPlot(treeOrd,trait,orderedChar = TRUE,cex = 0.3)
```

```
text(x = 41,y = 8,"ordered, biasStates = NULL",cex = 1.5)
quickAncPlot(treeOrdPrim,trait,orderedChar = TRUE,cex = 0.3)
text(x = 41.5,y = 8,"ordered, biasStates = 'primitive'",cex = 1.5)
quickAncPlot(treeOrdDer,trait,orderedChar = TRUE,cex = 0.3)
text(x = 42,y = 8,"ordered, biasStates = 'derived'",cex = 1.5)

#let's compare unordered with ordered, biasStates = 'primitive'
layout(1:2)
quickAncPlot(treeUnord,trait,orderedChar = FALSE,cex = 0.3)
text(x = 41,y = 8,"orderedChar = FALSE",cex = 1.5)
quickAncPlot(treeOrdPrim,trait,orderedChar = TRUE,cex = 0.3)
text(x = 40,y = 11,"orderedChar = TRUE",cex = 1.5)
text(x = 40,y = 4,"biasStates = 'primitive'",cex = 1.5)


#these comparisons will differ greatly between datasets
# need to try them on your own

layout(1)
```

`retiolitinae` *Cladogram and Range Data for the Retiolitinae*

Description

The majority rule consensus cladogram for 22 genera from the Retiolitinae, a clade of Silurian retiolitids, along with discrete time interval data taken from the same publication (Bates et al., 2005). Additional character state data are included for three major, binary-state morphological traits.

Format

This dataset is composed of three objects:

`retioTree` The consensus cladogram, given as an object of class `phylo`.

`retioRanges` A list containing two matrices. The first matrix describes the first and last interval times for 20 Silurian graptolite zones and the second matrix describes when the various genera on the cladogram first and last appear in those graptolite zones. (In other words, `retioRanges` has the `timeList` format called by some `paleotree` functions).

`retioChar` A matrix containing binary presence-absence character states for these 22 Retiolitinae genera for three characters which they vary in: the presence of extrathecal threads (note only one taxon lacks this character state), the presence of determinant growth and the secondary loss of a nema via resorption. Note these character do not vary within these genera.

Details

Interval dates were taken from Sadler et al. (2009). These zones were not a 1-1 match to those in Bates et al., so it took some merging and splitting by the package author, so buyer beware.

Character data are from an in prep manuscript containing character data for certain major morphological innovations of graptoloids, coded for a large number of genera based on an extensive survey of the published descriptions. The character data presented here is a small subset of the full dataset.

Source

Source for cladogram and zonal ranges for genera:

Bates, D. E. B., A. Kozlowska, and A. C. Lenz. 2005. Silurian retiolitid graptolites: Morphology and evolution. *Acta Palaeontologica Polonica* 50(4):705-720.

Source for interval dates for graptolite zones:

Sadler, P. M., R. A. Cooper, and M. Melchin. 2009. High-resolution, early Paleozoic (Ordovician-Silurian) time scales. *Geological Society of America Bulletin* 121(5-6):887-906.

Source for morphological character data:

Collected for Bapst and Mitchell, in prep.

See Also

For more example graptolite datasets, see `graptDisparity`

Examples

```
#load data
data(retiolitinae)

#Can plot discrete time interval diversity curve with retioRanges
taxicDivDisc(retioRanges)

#Can plot the unscaled cladogram
plot(retioTree)
#Can plot the determinant growth character on the cladogram
tiplabels(pch = 16, col = (retioChar[,2]+1),adj = 0.25)

#Use basic time-scaling (terminal branches only go to FADs)
ttree <- bin_timePaleoPhy(tree = retioTree,
    timeList = retioRanges,
    type = "basic",
   ntrees = 1,plot = TRUE)

#Note that this function creates stochastic time-scaled trees...
#A sample of 1 is not representative!

#phylogenetic diversity curve
phyloDiv(ttree)
```

`reverseList` *Reverse List Structure*

Description

Takes a list and reverses the list structure, such that list composed of five elements with eight sub-elements is restructured to have eight elements with five sub-elements each, with the order of elements and sub-elements being retained despite their reversal in hierarchical position.

Usage

```
reverseList(list, simplify = FALSE)
```

Arguments

`list`	A list composed of multiple elements, with each element a vector or list of equal length
`simplify`	Should the result be simplified, as with the identical argument in `sapply`?

Details

The function will fail and return an error if all sub-elements are not vectors or lists of equal length.

This function can be useful for instances when each element of a list is by-sample, composed of multiple, different tests on that sample, but where for further analysis/plotting, it would be beneficial to have a list where each element represented values from the same test performed across multiple samples (i.e. plotting a box-plot).

Value

Returns a list with a reversed structure relative to the input, see above.

Author(s)

David W. Bapst

Examples

```
list1 <- list(list(1:3),list(1:3),list(1:3))
reverseList(list1,simplify = FALSE)
reverseList(list1,simplify = TRUE)
```

rootSplit *Split Tip Taxa by Root Divergence*

Description

Sorts terminal taxa into groups descended from each lineage splitting off of the root node.

Usage

```
rootSplit(tree)
```

Arguments

tree	A phylogeny, as an object of class phylo.

Details

This function can be useful for studying the timing in the order of appearance of descended from different lineages descended from the first bifurcation.

Value

Returns a list with each element a character vector containing the names of terminal taxa descended from each lineage splitting off of the root node.

Author(s)

David W. Bapst

Examples

```
tree <- rtree(100)
rootSplit(tree)
```

sampleRanges *Sampling Taxon Ranges*

Description

A function for simulating the effect of incomplete sampling of the fossil record.

Usage

```
sampleRanges(
  taxaData,
  r,
  alpha = 1,
  beta = 1,
  rTimeRatio = 1,
  modern.samp.prob = 1,
  min.taxa = 2,
  ranges.only = TRUE,
  minInt = 0.01,
  merge.cryptic = TRUE,
  randLiveHat = TRUE,
  alt.method = FALSE,
  plot = FALSE
)
```

Arguments

taxaData	A two-column matrix of per-taxon ranges. The five-column matrix output of simFossilRecord, post transformation with fossilRecord2fossilTaxa can also be supplied, which will be common in simulation usages.
r	Instantaneous average sampling rate per lineage time units; given as a vector of length = 1 or length equal to the number of taxa.
alpha	Alpha parameter of beta distribution; given as a vector of length = 1 or length equal to the number of taxa.
beta	Beta parameter of beta distribution; given as a vector of length = 1 or length equal to the number of taxa.
rTimeRatio	Ratio of most recent sampling rate over earliest sampling rate; given as a vector of length = 1 or length equal to the number of taxa.
modern.samp.prob	Probability of sampling living taxa at the present day (time = 0), see below.
min.taxa	Minimum number of taxa sampled. The default is 2.
ranges.only	If TRUE, gives taxon first and last occurrences only. If FALSE, gives the time of all sampling events as a list.
minInt	Minimum interval size used for simulating complex models. See details.
merge.cryptic	If TRUE, sampling events for cryptic species will be merged into one taxon.
randLiveHat	If TRUE, taxa still alive at modern day have the end-point of their 'hat' chosen from a uniform distribution.
alt.method	If TRUE, use the alternative method of discretizing time even if a simple model of sampling is being simulated.
plot	If TRUE, plots the sampling models for each taxon against time.

Details

This function implements a range of sampling models in continuous time. Be default, sampling is simulated under the simplest model, where sampling occurs as a Poisson process under a instantaneous sampling rate (r) which is homogeneous through time and across lineages (Foote, 1997). Under this model, the waiting times to sampling events are exponentially distributed, with an average waiting time of 1/r. This useful property allows sampling to be rapidly simulated for many taxa under this simple model in sampleRanges, by repeatedly drawing waiting times between sampling events from an exponential distribution. This is the model that is run when alpha, beta and rTimeRatio.

In addition to this simple model, sampleRanges also can consider a range of additional models, including the *"hatP"* and *"incP"* options of Liow et al. (2010). To describe the behavior of these models, users alter the default values for alpha, beta and rTimeRatio. These parameters, and r, can either be a single value which describes the behavior of the entire dataset or as a vector, of same length as the number of taxa, which describes the per-taxon value. When any rTimeRatio, alpha or beta value is not equal to one, then the sampling rate will vary across the duration of a taxon's temporal range. In general, setting alpha and beta equal to a value above 2 will produce a "hat" or bell-shaped curve, where sampling rates peak at the midpoint of taxon ranges, while setting them unequal will produce asymmetric bell curves according to the beta function (Liow et al., 2010; Liow et al. set alpha = 4 and beta = 4). rTimeRatio is the ratio of the sampling rate of the latest (most recent) time divided by the earliest (oldest) time.

The input r values will be interpreted differently based on whether one r value or per-taxon values were used. If one value was input, then it is assumed that r represent the grand mean r for the entire dataset for purposes of time-varying r, such that if rTimeRatio is not equal to 1, taxa near the end and start of the dataset will have very different per-taxon mean sampling rate. If per-taxon values of r were input, then each r is consider the per-taxon mean sampling rate. These will not be changed, but any within-lineage variation is distributed so that the mean is still the input per-taxon value. This also changes the interpretation of rTimeRatio, such that when a single r value and rTimeRatio is given, it is assumed the ratio describes the change in sampling rates from the start of the dataset to the end, while if multiple values are given for either r or rTimeRatio will instead see the value as describing the ratio at the first and last times of each taxon. For the pure hat model, this interpretation of r as a grand mean sampling means that taxa will have a sampling rate of 2 * r at the mid-peak of their range, which will have considerable implications for taxonomic incompleteness.

The particular distinctions about these parameter values are important: all models simulated in sampleRanges are structured to be effectively nested inside a most general model with parameters r, alpha, beta and rTimeRatio.

Note that the modeling of sampling in this function is independent and secondary of the actual simulation of the ranges, which are (generally) produced by the models of simFossilRecord with argument r (sampling rate) not set. Thus, 'hat-shaped range distributions' are only contained within single morphotaxa – they do *not* cross multiple morphotaxa in the case of anagenesis. Cryptic taxa each have their own hat and do not share a single hat; by default the ranges of cryptic taxa are merged to produce the range of a single observed morphotaxon.

'Hats' are constrained to start and end with a taxon's range, representing the rise and fall of taxa in terms of abundance and geographic range (Liow et al., 2010). However, for still-living taxa at the modern day, it is unknown how much longer they may be alive (for memory-less Poisson models, there is no age-dependent extinction). The treatment of these taxa with regards to their 'hat' (i. e. the beta distribution) is controlled by the argument randLivehat. When randLiveHat = FALSE,

the beta distribution is fit so that the last appearance of still-alive taxa at the modern day is treated as a last appearance for calculating the hat. When TRUE, the default option, the still-alive taxa are considered to have gotten some distance between 0 and 1 through the beta distribution, as of the modern day. This point of progression is stochastically selected for each taxon by pulling a number from a uniform distribution, and used for calculating the hat.

Because sampling rate varies over morphotaxon ranges under any of these more complex models, sampling events cannot be quickly simulated as waiting times pulled from an exponential distribution. Instead, the taxon durations are discretized into a large number of small time intervals of length minInt (see above; minInt should be small enough that only one sampling event could feasibly happen per interval). The probability of an event occurring within each interval is calculated and used to stochastically simulate sampling events. For each interval, a number between 0 and 1 is randomly pulled from a uniform distribution and to the per-interval sampling probability to test if a sampling event occurred (if the random number is less than the probability, a sampling event is recorded). In general, this method is slower but otherwise comparable to the quicker waiting times method. See the examples below for a small test of this.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

If min.taxa is set to zero, the simulation may produce output in which no taxa were ever sampled.

If modern.samp.prob is set to 1.0 (the default), then living taxa will always be sampled at least at the present day (if there are any living taxa). If the probability is less than 1, they will be sampled with that probability at the modern day.

By default, this function will merge sampling events from morphologically cryptic taxa, listing them as occurrences for the earliest member of that group. To change this behavior, set merge.cryptic = FALSE.

Conditioning on sampling some minimum number of taxa may create strange simulation results for some analyses, such as simulation analyses of birth-death processes. Set min.taxa = 0 to remove this conditioning.

Value

If ranges.only is TRUE, then the output is a two-column per-taxon matrix of first and last appearances in absolute time. NAs mean the respective taxon was never sampled in the simulation.

If ranges.only = FALSE (the default), the output is a list, where each element is a vector of sampling events the timing of sampling events, each corresponding to a different taxon in the input. Elements that are NA are unsampled taxa.

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Liow, L. H., T. B. Quental, and C. R. Marshall. 2010 When Can Decreasing Diversification Rates Be Detected with Molecular Phylogenies and the Fossil Record? *Systematic Biology* **59**(6):646–659.

See Also

simFossilRecord, binTimeData

Examples

```
set.seed(444)
record <- simFossilRecord(
             p = 0.1,
             q = 0.1,
             nruns = 1,
            nTotalTaxa = c(30,40),
            nExtant = 0
            )
taxa <- fossilRecord2fossilTaxa(record)

# let's see what the 'true' diversity curve looks like in this case
layout(1:2)
taxicDivCont(taxa)
# simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r = 0.5)
# plot the diversity curve based on the sampled ranges
taxicDivCont(rangesCont)

# compare the true history to what we might observe!

#let's try more complicated models!

# a pull-to-the-recent model with x5 increase over time
   # similar to Liow et al.'s  incP
layout(1:2)
rangesCont1 <- sampleRanges(taxa,
                            r = 0.5,
                            rTimeRatio = 5,
                            plot = TRUE
                            )
taxicDivCont(rangesCont1)

# a hat-shaped model
layout(1:2)
rangesCont1 <- sampleRanges(taxa,
                            r = 0.5,
                            alpha = 4,
                            beta = 4,
                            plot = TRUE
                            )
taxicDivCont(rangesCont1)

# a combination of these
layout(1:2)
rangesCont1 <- sampleRanges(taxa,
                            r = 0.5,
```

```
                            alpha = 4,
                            beta = 4,
                            rTimeRatio = 5,
                            plot = TRUE
                            )
taxicDivCont(rangesCont1)

# testing with cryptic speciation
layout(1)
recordCrypt <- simFossilRecord(p = 0.1, q = 0.1,
                               prop.cryptic = 0.5,
                               nruns = 1,
                               nTotalTaxa = c(20,30),
                               nExtant = 0
                               )
taxaCrypt <- fossilRecord2fossilTaxa(recordCrypt)
rangesCrypt <- sampleRanges(taxaCrypt,r = 0.5)
taxicDivCont(rangesCrypt)

#an example of hat-shaped models (beta distributions) when there are live taxa
set.seed(444)
recordLive <- simFossilRecord(p = 0.1,
                              q = 0.05,
                              nruns = 1,
                              nTotalTaxa = c(5,100),
                              nExtant = c(10,100)
                              )
taxaLive <- fossilRecord2fossilTaxa(recordLive)
#with end-points of live taxa at random points in the hat
rangesLive <- sampleRanges(taxaLive,
                           r = 0.1,
                           alpha = 4,
                           beta = 4,
                           randLiveHat = TRUE,
                           plot = TRUE
                           )
#with all taxa end-points at end-point of hat
rangesLive <- sampleRanges(taxaLive,
                           r = 0.1,
                           alpha = 4,
                           beta = 4,
                           randLiveHat = FALSE,
                           plot = TRUE
                           )



#simulate a model where sampling rate evolves under brownian motion
tree <- taxa2phylo(taxa,obs = taxa[,3])
sampRateBM <- rTraitCont(tree)
sampRateBM <- sampRateBM-min(sampRateBM)
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r = sampRateBM,plot = TRUE)
```

```
taxicDivCont(rangesCont1)

#evolving sampling rate, hat model and pull of the recent
layout(1:2)
rangesCont1 <- sampleRanges(taxa,
                            r = sampRateBM,
                            alpha = 4,
                            beta = 4,
                            rTimeRatio = 5,
                            plot = TRUE
                            )
taxicDivCont(rangesCont1)
layout(1)

#the simpler model is simulated by pulling waiting times from an exponential
#more complicated models are simulated by discretizing time into small intervals
#are these two methods comparable?

#let's look at the number of taxa sampled under both methods

summary(replicate(100,sum(!is.na(
               sampleRanges(taxa,
                            r = 0.5,
                            alt.method = FALSE
                            )
               [,1]))))

summary(replicate(100,sum(!is.na(
               sampleRanges(taxa,
                            r = 0.5,
                            alt.method = TRUE
                            )
               [,1]))))

#they look pretty similar!
```

SamplingConv *Converting Sampling Estimates*

Description

Various functions for converting between estimates of sampling in the fossil record.

Usage

```
sProb2sRate(R, int.length = 1)

sRate2sProb(r, int.length = 1)
```

```
pqsRate2sProb(r, p, q, int.length = 1)

qsProb2Comp(R, q, p = NULL, mode = "budding", nrep = 10000)

qsRate2Comp(r, q)
```

Arguments

R	Per-interval probability of sampling a taxon at least once.
int.length	Length of Time Intervals
r	Instantaneous rate of sampling (per taxon, per time-unit).
p	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.
q	Instantaneous rate of extinction (mu)
mode	Mode of morphotaxon differentiation, based on definitions in Foote, 1996. Can be pure cladogenetic budding ("budding"), pure cladogenetic bifurcating ("bifurcating") or pure anagenetic within-lineage change ("anagenesis"; i.e. Foote's 'phyletic change'). Default mode is "budding".
nrep	Number of repetitions to run in functions which are meant to sum over infinity. Default is arbitrarily high.

Details

This is a family of functions which all convert from some estimate of sampling to another estimate of sampling. Some of these also require estimates of an rate associated with taxonomic diversification, such as the speciation (or origination) rate or extinction rate. Diversification rates used in these functions should always be the instantaneous rates, often called the per-capita rates by paleontologists (Foote, 2000).

As with many models used in the paleotree library, it is generally assumed by these functions that the fossil record of interest is composed of discrete relatively-static taxonomic units which diversify typically by budding cladogenesis, and that sampling events are rare and approximated by a Poisson model of exponentially-distributed waiting times between sampling events. The veracity of those assumptions is difficult to test and the sensitivity of these analyses to relaxing those assumptions probably varies.

sProb2sRate and sRate2sProb give rough conversions for the probability of sampling once per time interval (the variable R or sProb in this package as used in the references below) and the instantaneous rate of sampling per lineage/time unit (sRate or r). If you have estimates of the speciation and extinction rate, use pqsRate2sProb instead for a more accurate estimate of R.

qsProb2Comp and qsRate2Comp are different calculations for the probability/proportion of taxa sampled in a clade (often labeled as the variable Pp). Theoretically, one could use it to extrapolate out the 'true' diversity, assuming the sampling rate model was correct. (See Foote and Raup, 1996.)

See the references below for a more detailed explanation of the methods and formulae used. The relevant equations are generally found in the appendices of those papers.

Value

The converted sampling estimate, depending on the function used. See details above.

Author(s)

David W. Bapst, with advice from Michael Foote.

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M. 2000 Origination and extinction components of taxonomic diversity: general problems. Pp. 74–102. In D. H. Erwin, and S. L. Wing, eds. Deep Time: Paleobiology's Perspective. The Paleontological Society, Lawrence, Kansas.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

Solow, A. R., and W. Smith. 1997 On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* **23**(3):271–277.

See Also

`sampleRanges`, `make_durationFreqDisc`, `make_durationFreqCont`, `probAnc`, `pqr2Ps`.

Examples

```
sRate2sProb(r = 0.5)
sProb2sRate(R = 0.1)
pqsRate2sProb(r = 0.5,p = 0.1,q = 0.1)

# different modes can be tried
qsProb2Comp(R = 0.1,q = 0.1,mode = "budding")
qsProb2Comp(R = 0.1,q = 0.1,mode = "bifurcating")

qsRate2Comp(r = 0.1,q = 0.1)
```

---

seqTimeList — *Construct a Stochastic Sequenced Time-List from an Unsequenced Time-List*

Description

This function randomly samples from a `timeList` object (i.e. a list composed of a matrix of interval start and end dates and a matrix of taxon first and last intervals), to find a set of taxa and intervals that do not overlap, output as a new `timeList` object.

Usage

```
seqTimeList(timeList, nruns = 100, weightSampling = FALSE)
```

Arguments

timeList	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. Some intervals are expected to overlap (thus necessitating the use of this function), and datasets lacking overlapping intervals will return an error message.
nruns	Number of new timeList composed of non-overlapping intervals produced.
weightSampling	If TRUE, weight sampling of new intervals toward smaller intervals. FALSE by default.

Details

Many analyses of diversification and sampling in the fossil record require a dataset composed of sequential non-overlapping intervals, but the nature of the geologic record often makes this difficult, with taxa from different regions, environments and sedimentary basins having first and last appearances placed in entirely in-congruent systems of chronostratigraphic intervals. While one option is to convert such occurrences to a single, global stratigraphic system, this may still result in overlapping intervals when fossil collections are poorly constrained stratigraphically. (For example, this may often be the case in global datasets.) This function offers an approach to avoid this issue in large datasets by randomly subsampling the available taxa and intervals to produce stochastic sets of ranges composed of data drawn from non-overlapping intervals.

seqTimeList is stochastic and thus should be set for many runs to produce many such solutions. Additionally, all solutions found are returned, and users may wish to sort amongst these to maximize the number of intervals and number of taxa returned. A single solution which maximizes returned taxa and intervals may not be a precise enough approach to estimating sampling rates, however, given the uncertainty in data. Thus, many runs should always be considered.

By default, solutions are searched for without consideration to the length of intervals used (i.e. the selection of intervals is 'unweighted'). Alternatively, we can 'weight' selection toward the smallest intervals in the set, using the argument weightSampling. Smaller intervals presumably overlap less and thus should retain more taxa and intervals of more equal length. However, in practice with empirical datasets, the package author finds these approaches do not seem to produce very different estimates.

For some datasets, many solutions found using seqTimeList may return infinite sampling values. This is often due to saving too many taxa found in single intervals to the exclusion of longer-ranging taxa (see the example). This excess of single interval taxa is a clear artifact of the randomized seqTimeList procedure and such solutions should probably be ignored.

Value

A list, composed of three elements: nIntervals, a vector of the number of intervals in each solution, nTaxa, a vector of the number of taxa in each solution, and timeLists, a list composed of each new timeList object as an element.

Author(s)

David W. Bapst

See Also

Resulting time-lists can be analyzed with `freqRat`, `durationFreq`, etc.

Additionally, `binTimeData` can be useful for simulating interval data.

Examples

```
# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    nruns = 1,
    nTotalTaxa = c(60,80),
    nExtant = 0
    )
taxa <- fossilRecord2fossilTaxa(record)
# simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r = 0.1)

# Now let's use binTimeData to get ranges in discrete overlapping intervals
    # via pre-set intervals input
presetIntervals <- cbind(
  c(1000, 995, 990, 980, 970, 975, 960, 950, 940, 930, 900, 890, 888, 879, 875),
   c(995, 989, 960, 975, 960, 950, 930, 930, 930, 900, 895, 888, 880, 875, 870)
  )
rangesDisc1 <- binTimeData(rangesCont, int.times = presetIntervals)

seqLists <- seqTimeList(rangesDisc1, nruns = 10)
seqLists$nTaxa
seqLists$nIntervals

#apply freqRat as an example analysis
sapply(seqLists$timeLists, freqRat)

# notice the zero and infinite freqRat estimates? What's going on?

freqRat(seqLists$timeLists[[4]], plot = TRUE)

# too few taxa of two or three interval durations for the ratio to work properly
    # perhaps ignore these estimates

# with weighted selection of intervals
seqLists <- seqTimeList(rangesDisc1, nruns = 10, weightSampling = TRUE)

seqLists$nTaxa
seqLists$nIntervals
sapply(seqLists$timeLists, freqRat)
```

```
# didn't have much effect in this simulated example
```

`setRootAge` *Place a Non-Ultrametric Tree of Fossil Taxa on Absolute Time*

Description

This function uses a table of fixed dates for operational-taxon-units (tip taxa) to calculate the absolute age of the root divergence for a tree with branch lengths, and then appends this root age to the tree as a `$root.time` element, and then outputs the tree. Function `setRootAges` is a wrapper for `setRootAge` for use with multiple trees in a object of class `multiPhylo`. This function was mainly written for dealing with trees of extinct taxa dated in units of absolute time from Bayesian analyses, such as with MrBayes, with trees scaled to time units by functions such as `obtainDatedPosteriorTreesMrB`.

Usage

```
setRootAge(tree, fixedAges = NULL)

setRootAges(trees, fixedAges = NULL)
```

Arguments

`tree`	A phylogeny with branch lengths of class `phylo`.
`fixedAges`	A table of fixed ages for tip taxa, generally as a dataframe where the first column is of type character, and the second column is of type `numeric`. Such a table is automatically generated as an attribute of the output from `obtainDatedPosteriorTreesMrB`, when argument `getFixedTimes = TRUE`.
`trees`	A list of class `multiPhylo` consisting of multiple phylogenetic trees with branch lengths.

Details

Trees of fossil taxa come with one issue rarely encountered by those dealing with molecular phylogenies: the absolute timing of when tips and divergences is not certain. With the vast majority of molecular phylogenies, it can be assumed the youngest tips occur at time 0 – in other words , the modern. This knowledge gives the tree an *'anchor'* for fixing the absolute timing of events. Many programs and other software designed for depicting and analyzing phylogenetic hypotheses assumes such an apparent absolute time-scale (in R and elsewhere). A phylogenetic analysis of Paleozoic brachiopods that include no extant members has no such anchor at time = 0, and such a default assumption in available software can be misleading. The `$root.time` protocol is intended to grant this absolute time-scale to a dated tree of fossil taxa, and is appended by most of the dating functions in package paleotree. However, trees dated by other approaches, such as via tip-dating in programs such as MrBayes and BEAST, will not have `$root.time` elements when read into R.

Value

The input tree is output, with a new `$root.time` element.

Author(s)

David W. Bapst

See Also

`setRootAges` is designed to work by default with trees on relative time-scales dated by `obtainDatedPosteriorTreesMrB`, particularly when the argument with `getFixedTimes = TRUE`, which is used to obtain fixed tip ages for anchoring the tree against an absolute time-scale. The functions described here will be applied automatically with `obtainDatedPosteriorTreesMrB` if argument `getRootAges = TRUE`.

Examples

```
set.seed(444)
tree <- rtree(10)
tipAges <- cbind(c("t1","t2"), c(15,10))

absTimeTree <- setRootAge(tree = tree,tipAges)

plot(absTimeTree)
axisPhylo()
```

---

simFossilRecord *Full-Scale Simulations of the Fossil Record with Birth, Death and Sampling of Morphotaxa*

Description

A complete birth-death-sampling branching simulator that captures morphological-taxon identity of lineages, as is typically discussed in models of paleontological data. This function allows for the use of precise point constraints to condition simulation run acceptance and can interpret complex character strings given as rate values for use in modeling complex processes of diversification and sampling.

Usage

```
simFossilRecord(
  p,
  q,
  r = 0,
  anag.rate = 0,
  prop.bifurc = 0,
  prop.cryptic = 0,
  modern.samp.prob = 1,
  startTaxa = 1,
  nruns = 1,
  maxAttempts = Inf,
```

```
  totalTime = c(0, 1000),
  nTotalTaxa = c(1, 1000),
  nExtant = c(0, 1000),
  nSamp = c(0, 1000),
  returnAllRuns = FALSE,
  tolerance = 10^-6,
  maxStepTime = 0.01,
  shiftRoot4TimeSlice = "withExtantOnly",
  count.cryptic = FALSE,
  negRatesAsZero = TRUE,
  print.runs = FALSE,
  sortNames = FALSE,
  plot = FALSE
)
```

Arguments

`p, q, r, anag.rate`

These parameters control the instantaneous ('per-capita') rates of branching, extinction, sampling and anagenesis, respectively. These can be given as a number equal to or greater than zero, or as a character string which will be interpreted as an algebraic equation. These equations can make use of three quantities which will/may change throughout the simulation: the standing richness is `N`, the current time passed since the start of the simulation is `T`, the present duration of a given still-living lineage since its origination time is codeD, and the current branching rate is `P` (corresponding to the argument name `p`). Note that `P` cannot be used in equations for the branching rate itself; it is for making other rates relative to the branching rate.

By default, the rates `r` and `anag.rate` are set to zero, so that the default simulator is a birth-death simulator. Rates set to `= Inf` are treated as if 0. When a rate is set to 0, this event type will not occur in the simulation. Setting certain processes to zero, like sampling, may increase simulation efficiency, if the goal is a birth-death or pure-birth model. See documentation for argument `negRatesAsZero` about the treatment of rates that decrease below zero. Notation of branching, extinction and sampling rates as `p,q,r` follows what is typical for the paleobiology literature (e.g. Foote, 1997), not the Greek letters `lambda,mu,phi` found more typically in the biological literature (e.g. Stadler, 2009; Heath et al., 2014; Gavryushkina et al., 2014).

`prop.cryptic, prop.bifurc`

These parameters control (respectively) the proportion of branching events that have morphological differentiation, versus those that are cryptic (`prop.cryptic`) and the proportion of morphological branching events that are bifurcating, as opposed to budding. Both of these proportions must be a number between 0 and 1. By default, both are set to zero, meaning all branching events are events of budding cladogenesis. See description of the available models of morphological differentiation in the *Description* section.

`modern.samp.prob`

The probability that a taxon is sampled at the modern time (or, for `timeSliceFossilRecord`, the time at which the simulation data is slice). Must be a number between 0 and

1. If 1, all taxa that survive to the modern day (to the sliceTime) are sampled, if 0, none are.

startTaxa — Number of initial taxa to begin a simulation with. All will have the simulation start date listed as their time of origination.

nruns — Number of simulation datasets to accept, save and output. If nruns = 1, output will be a single object of class fossilRecordSimulation, and if nruns is greater than 1, a list will be output composed of nruns objects of class fossilRecordSimulation.

maxAttempts — Number of simulation attempts allowed before the simulation process is halted with an error message. Default is Inf.

totalTime, nTotalTaxa, nExtant, nSamp — These arguments represent stopping and acceptance conditions for simulation runs. They are respectively totalTime, the total length of the simulation in time-units, nTotalTaxa, the total number of taxa over the past evolutionary history of the clade, nExtant, the total number of extant taxa at the end of the simulation and nSamp the total number of sampled taxa (not counting extant taxa sampled at the modern day). These are used to determine when to end simulation runs, and whether to accept or reject them as output. They can be input as a vector of two numbers, representing minimum and maximum values of a range for accepted simulation runs (i.e. the simulation length can be between 0 and 1000 time-steps, by default), or as a single number, representing a point condition (i.e. if nSamp = 100 then only those simulation states that contain exactly 100 taxa sampled will be output). Note that it is easy to set combinations of parameters and run conditions that are impossible to produce satisfactory input under, in which case simFossilRecord would run in a nonstop loop. How cryptic taxa are counted for the sake of these conditions is controlled by argument count.cryptic. Note that behavior of these constraints can be modified by the argument returnAllRuns.

returnAllRuns — If TRUE, all simulation runs will be returned, with the output given as a list composed of two sublists - the first sublist containing all accepted simulations (i.e. everything that would be returned under the default condition of returnAllRuns = FALSE), and the second sublist containing the full history of each failed simulation. These failed simulations are only stopped when they one of four, irreversible 'out-of-bounds' constraints. These four conditions are (a) reaching the maximum total simulation duration (totalTime), (b) exceeding the maximum number of total taxa (nTotalTaxa), (c) exceeding the maximum number of sampled taxa (nSamp), or (d) total extinction of all lineages in the simulation.

tolerance — A small number which defines a tiny interval for the sake of placing run-sampling dates before events and for use in determining whether a taxon is extant in simFossilRecordMethods.

maxStepTime — When rates are time-dependent (i.e. when parameters 'D' or 'T' are used in equations input for one of the four rate arguments), then protocol used by simFossilRecord of drawing waiting times to the next event could produce a serious mismatch of resulting process to the defined model, because the possibility of new events is only considered at the end of these waiting times. Instead, any time a waiting time greater than maxStepTime is selected, then instead *no* event occurs and a time-step equal to maxStepTime occurs instead, thus effectively discretizing

the progression of time in the simulations run by simFossilRecord. Decreasing this value will increase accuracy (as the time-scale is effectively more discretized) but increase computation time, as the computer will need to stop and check rates to see if an event happened more often. Users should toggle this value relative to the time-dependent rate equations they input, relative to the rate of change in rates expected in time-dependent rates.

shiftRoot4TimeSlice
: Should the dating of events be shifted, so that the date given for sliceTime is now 0, or should the dates not be shifted, so that they remain on the same scale as the input? This argument accepts a logical TRUE or FALSE, but also accepts the string "withExtantOnly", which will only 'shift' the time-scale if living taxa are present, as determined by having ranges that overlap within tolerance of sliceTime.

count.cryptic
: If TRUE, cryptic taxa are counted as separate taxa for conditioning limits that count a number of taxon units, such as nTotalTaxa, nExtant and nSamp. If FALSE (the default), then each cryptic complex (i.e. each distinguishable morphotaxon) is treated as a single taxon. See examples.

negRatesAsZero
: A logical. Should rates calculated as a negative number cause the simulation to fail with an error message (= FALSE) or should these be treated as zero (= TRUE, the default). This is equivalent to saying that the rate.as.used = max(0,rate.as.given).

print.runs
: If TRUE, prints the proportion of simulations accepted for output to the terminal.

sortNames
: If TRUE, output taxonomic lists are sorted by the taxon names (thus sorting cryptic taxa together) rather than by taxon ID number (i.e. the order they were simulated in).

plot
: If TRUE, plots the diversity curves of accepted simulations, including both the diversity curve of the true number of taxa and the diversity curve for the 'observed' (sampled) number of taxa.

Details

simFossilRecord simulates a birth-death-sampling branching process (ala Foote, 1997, 2000; Stadler, 2009) in which lineages of organisms may branch, go extinct or be sampled at discrete points within a continuous time-interval. The occurrence of these discrete events are modeled as stochastic Poisson process, described by some set of instantaneous rates. This model is ultimately based on the birth-death model (Kendall, 1948; Nee, 2006), which is widely implemented in many R packages. Unlike other such typical branching simulators, this function enmeshes the lineage units within explicit models of how lineages are morphologically differentiated (Bapst, 2013). This is key to allow comparison to datasets from the fossil record, as morphotaxa are the basic units of paleontological diversity estimates and phylogenetic analyses.

Models of Morphological Differentiation and Branching (Cladogenesis and Anagenesis)

These models of morphological differentiation do not involve the direct simulation of morphological traits. Instead, morphotaxon identity is used as a proxy of the distinctiveness of lineages on morphological grounds, as if there was some hypothetical paleontologist attempting to taxonomically sort collections of specimens of these simulated lineages. Two lineages are either identical,

and thus share the same morphotaxon identity, or they are distinct, and thus have separate morphotaxon identities. Morphological differentiation is assumed to be an instantaneous process for the purposes of this model, such that no intermediate could be uncovered.

Specifically, `simFossilRecord` allows for three types of binary branching events, referred to here as under the umbrella term of 'cladogenesis': 'budding cladogenesis', 'bifurcating cladogenesis', and 'cryptic cladogenesis', as well as for a fourth non-branching event-type, 'anagenesis'. See Wagner and Erwin, 1995; Foote, 1996; and Bapst, 2013, for further details. Budding, bifurcation and cryptic cladogenetic events all share in common that a single geneological lineage splits into two descendant lineages, but differ in the morphological differentiation of these child lineages relative to their parent. Under budding cladogenesis, only one of the child lineages becomes morphologically distinguishable from the parent, and thus the ancestral morphotaxon persists through the branching event as the child lineage that does *not* differentiate. Under bifurcating cladogenesis, both child lineages become immediately distinct from the ancestor, and thus two new morphotaxa appear while the ancestor terminates in an event known as 'pseudoextinction'. Cryptic cladogenesis has no morphological differentiation: both child lineages are presumed to be indistinct from the ancestor and from each other, which means a hypothetical paleontologist would not observe that branching had occurred at all. Anagenesis is morphological differentiation independent of any branching, such that a morphotaxon instantaneously transitions to a new morphotaxon identity, resulting in the pseudoextinction of the ancestral morphotaxon and the immediate 'pseudospeciation' of the child morphotaxon. In anagenesis, the ancestral morphotaxon and descendant morphotaxon do not overlap in time at all, as modeled here (contra to the models described by Ezard et al., 2012). For ease of following these cryptic lineages, cryptic cladogenetic events are treated in terms of data structure similarly to budding cladogenetic events, with one child lineage treated as a persistence of the ancestral lineage, and the other as a new morphologically indistinguishable lineage. This model of cryptic cladogenesis is ultimately based on the hierarchical birth-death model used by many authors for modeling patterns across paraphyletic higher taxa and the lower taxon units within them (e.g. Patzkowsky, 1995; Foote, 2012).

The occurrence of the various models is controlled by multiple arguments of `simFossilRecord`. The overall instantaneous rate of branching (cladogenesis) is controlled by argument `p`, and the proportion of each type of cladogenesis controlled by arguments `prop.bifurc` and `prop.cryptic`. `prop.cryptic` controls the overall probability that any branching event will be cryptic versus involving any morphological differentiation (budding or bifurcating). If `prop.cryptic = 1`, all branching events will be cryptic cladogenesis, and if `prop.cryptic = 0`, all branching events will involve morphological differentiation and none will be cryptic. `prop.bifurc` controls how many branching events that involve morphological differentiation (i.e. the inverse of `prop.cryptic`) are bifurcating, as opposed to budding cladogenesis. If `prop.bifurc = 1`, all morphologically-differentiating branching events will be bifurcating cladogenesis, and if `prop.bifurc = 0`, all morphologically-differentiating branching events will be budding cladogenesis. Thus, for example, the probability of a given cladogenesis event being budding is given by:

`Prob(budding cladogenesis at a branching event) = (1 -prop.cryptic) * (1 -prop.bifurc)`

By default, `prop.cryptic = 0` and `prop.bifurc = 0`, so all branching events will be instances of budding cladogenesis in analyses that use default setting. Anagenesis is completely independent of these, controlled as its own Poisson process with an instantaneous rated defined by the argument `anag.rate`. By default, this rate is set to zero and thus there is no anagenetic events without user intervention.

Stopping Conditions and Acceptance Criteria for Simulations

How forward-time simulations are generated, halted and whether they are accepted or not for output

is a critical component of simulation design. Most uses of simFossilRecord will involve iteratively generating and analyzing multiple simulation runs. Runs are only accepted for output if they meet the conditioning criteria defined in the arguments, either matching point constraints or falling within range constraints. However, this requires separating the processes of halting simulation runs and accepting a run for output, particularly to avoid bias related to statistical sampling issues.

Hartmann et al. (2011) recently discovered a potential statistical artifact when branching simulations are conditioned on some number of taxa. Previously within paleotree, this was accounted for in the deprecated function simFossilTaxa by a complex arrangement of minimum and maximum constraints, and an (incorrect) presumption that allowing simulations to continue for a short distance after constraints were reached would solve this statistical artifact. This strategy is not applied here. Instead, simFossilRecord applies the General Sampling Algorithm presented by Hartmann et al. (or at least, a close variant). A simulation continues until extinction or some maximum time-constraint is reached, evaluated for intervals that match the set run conditions (e.g. nExtant, nTotalTime) and, if some interval or set of intervals matches the run conditions, a date is randomly sampled from within this interval/intervals. The simulation is then cut at this date using the timeSliceFossilRecord function, and saved as an accepted run. The simulation data is otherwise discarded and then a new simulation initiated (therefore, at most, only one simulated dataset is accepted from one simulation run).

Thus, accepted simulations runs should reflect unbiased samples of evolutionary histories that precisely match the input constraints, which can be very precise, unlike how stopping and acceptance conditions were handled in the previous (deprecated) simFossilTaxa function. Of course, selecting very precise constraints that are very unlikely or impossible given other model parameters may take considerable computation time to find acceptable simulation runs, or effectively never find any acceptable simulation runs.

On Time-Scale Used in Output

Dates given in the output are on an reversed absolute time-scale; i.e. time decreases going from the past to the future, as is typical in paleontological uses of time (as time before present) and as for most function in package paleotree. The endpoints of the time-scale are decided by details of the simulation and can be modified by several arguments. By default (with shiftRoot4TimeSlice = "withExtantOnly"), any simulation run that is accepted with extant taxa will have zero as the *end-time* (i.e. when those taxa are extant), as zero is the typical time assigned to the modern day in empirical studies. If a simulation ends with all taxa extinct, however, then instead the *start-time* of a run (i.e. when the run initiates with starting taxa) will be maximum value assigned to the conditioning argument totalTime. If shiftRoot4TimeSlice = FALSE, then the *start-time* of the run will always be this maximum value for totalTime, and any extant taxa will stop at some time greater than zero.

Value

simFossilRecord returns either a single object of class fossilRecordSimulation or a list of multiple such objects, depending on whether nruns was 1 or more. If argument returnAllRuns = TRUE, a list composed of two sublists, each of which contains 0 or more fossilRecordSimulation objects. The first sublist containing all the accepted simulations (i.e. all the simulations that would have been returned if returnAllRuns was FALSE), and the second sublist containing the final iteration of all rejected runs before they hit an irreversible out-of-bounds condition (to wit, reaching the maximum totalTime, exceeding the maximum number of total taxa (nTotalTaxa), exceeding the maximum number of sampled taxa (nSamp), or total extinction of all lineages in the simulation).

An object of class `fossilRecordSimulation` consists of a list object composed of multiple elements, each of which is data for 'one taxon'. Each data element for each taxon is itself a list, composed of two elements: the first describes vital information about the taxon unit, and the second describes the sampling times of each taxon.

The first element of the list (named `$taxa.data`) is a distinctive six-element vector composed of numbers (some are nominally integers, but not all, so all are stored as double-precision integers) with the following field names:

`taxon.id` The ID number of this particular taxon-unit.

`ancestor.id` The ID number of the ancestral taxon-unit. The initial taxa in a simulation will be listed with `NA` as their ancestor.

`orig.time` True time of origination for a taxon-unit in absolute time.

`ext.time` True time of extinction for a taxon-unit in absolute time. Extant taxa will be listed with an `ext.time` of the run-end time of the simulation run, which for simulations with extant taxa is 0 by default (but this may be modified using argument `shiftRoot4TimeSlice`).

`still.alive` Indicates whether a taxon-unit is 'still alive' or not: '1' indicates the taxon-unit is extant, '0' indicates the taxon-unit is extinct

`looks.like` The ID number of the first morphotaxon in a dataset that 'looks like' this taxon-unit; i.e. belongs to the same multi-lineage cryptic complex. Taxa that are morphologically distinct from any previous lineage will have their `taxon.id` match their `looks.like`. Thus, this column is rather uninformative unless cryptic cladogenesis occurred in a simulation.

The second element for each taxon-unit is a vector of sampling times, creatively named `$sampling.times`, with each value representing a data in absolute time when that taxon was sampled in the simulated fossil record. If a taxon was never sampled, this vector is an empty numeric vector of `length = 0`.

As is typical for paleontological uses of absolute time, absolute time in these simulations is always decreasing toward the modern; i.e. an absolute date of 50 means a point in time which is 50 time-units before the present-day, if the present-day is zero (the default, but see argument `shiftRoot4TimeSlice`).

Each individual element of a `fossilRecordSimulation` list object is named, generally of the form `"t1"` and `"t2"`, where the number is the `taxon.id`. Cryptic taxa are instead named in the form of `"t1.2"` and `"t5.3"`, where the first number is the taxon which they are a cryptic descendant of (`looks.like`) and the second number, after the period, is the order of appearance of lineage units in that cryptic complex. For example, for `"t5.3"`, the first number is the `taxon.id` and the second number communicates that this is the third lineage to appear in this cryptic complex.

Author(s)

David W. Bapst, inspired by code written by Peter Smits.

References

Bapst, D. W. 2013. When Can Clades Be Potentially Resolved with Morphology? *PLoS ONE* 8(4):e62312.

Ezard, T. H. G., P. N. Pearson, T. Aze, and A. Purvis. 2012. The meaning of birth and death (in macroevolutionary birth-death models). *Biology Letters* 8(1):139-142.

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

Foote, M. 1997. Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* 23(3):278-300.

Foote, M. 2000. Origination and extinction components of taxonomic diversity: general problems. Pp. 74-102. In D. H. Erwin, and S. L. Wing, eds. *Deep Time: Paleobiology's Perspective.* The Paleontological Society, Lawrence, Kansas.

Foote, M. 2012. Evolutionary dynamics of taxonomic structure. *Biology Letters* 8(1):135-138.

Gavryushkina, A., D. Welch, T. Stadler, and A. J. Drummond. 2014. Bayesian Inference of Sampled Ancestor Trees for Epidemiology and Fossil Calibration. *PLoS Computational Biology* 10(12):e1003919.

Hartmann, K., D. Wong, and T. Stadler. 2010 Sampling Trees from Evolutionary Models. *Systematic Biology* **59**(4):465–476.

Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences* 111(29):E2957-E2966.

Kendall, D. G. 1948 On the Generalized "Birth-and-Death" Process. *The Annals of Mathematical Statistics* **19**(1):1–15.

Nee, S. 2006 Birth-Death Models in Macroevolution. *Annual Review of Ecology, Evolution, and Systematics* **37**(1):1–17.

Patzkowsky, M. E. 1995. A Hierarchical Branching Model of Evolutionary Radiations. *Paleobiology* 21(4):440-460.

Solow, A. R., and W. Smith. 1997 On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* **23**(3):271–277.

Stadler, T. 2009. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of Theoretical Biology* 261(1):58-66.

Wagner, P. J., and D. H. Erwin. 1995. Phylogenetic patterns as tests of speciation models. Pp. 87-122. In D. H. Erwin, and R. L. Anstey, eds. *New approaches to speciation in the fossil record.* Columbia University Press, New York.

See Also

`simFossilRecordMethods`

This function essentially replaces and adds to all functionality of the deprecated `paleotree` functions `simFossilTaxa`, `simFossilTaxaSRCond`, `simPaleoTrees`, as well as the combined used of `simFossilTaxa` and `sampleRanges` for most models of sampling.

Examples

```
set.seed(2)

# quick birth-death-sampling run
    # with 1 run, 50 taxa

record <- simFossilRecord(
```

```
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    nTotalTaxa = 50,
    plot = TRUE
    )

################


# Now let's examine with multiple runs of simulations

# example of repeated pure birth simulations over 50 time-units
records <- simFossilRecord(
    p = 0.1,
    q = 0,
    nruns = 10,
    totalTime = 50,
    plot = TRUE
    )

# plot multiple diversity curves on a log scale
records <- lapply(records,
    fossilRecord2fossilTaxa)
multiDiv(records,
    plotMultCurves = TRUE,
    plotLogRich = TRUE
    )

# histogram of total number of taxa
hist(sapply(records, nrow))

##############################################
# example of repeated birth-death-sampling
    # simulations over 50 time-units

records <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 10,
    totalTime = 50,
    plot = TRUE)

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
    plotMultCurves = TRUE)

# like above...
    # but conditioned instead on having 10 extant taxa
```

```
    # between 1 and 100 time-units

set.seed(4)

records <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 10,
    totalTime = c(1,300),
    nExtant = 10,
    plot = TRUE
    )

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
    plotMultCurves = TRUE
    )

################################################

# How probable were the runs I accepted?
    # The effect of conditions

set.seed(1)

# Let's look at an example of a birth-death process
    # with high extinction relative to branching

# notes:
    # a) use default run conditions (barely any conditioning)
    # b) use print.runs to look at acceptance probability

records <- simFossilRecord(
    p = 0.1,
    q = 0.8,
    nruns = 10,
    print.runs = TRUE,
    plot = TRUE
    )

# 10 runs accepted from a total of 10 !

# now let's give much more stringent run conditions
    # require 3 extant taxa at minimum, 5 taxa total minimum

records <- simFossilRecord(
    p = 0.1,
    q = 0.8,
    nruns = 10,
    nExtant = c(3,100),
```

```
    nTotalTaxa = c(5,100),
    print.runs = TRUE,
    plot = TRUE
    )

# thousands of simulations to just obtail 10 accepable runs!
    # most ended in extinction before minimums were hit

# beware analysis of simulated where acceptance conditions
    # are too stringent: your data will be a 'special case'
    # of the simulation parameters
# it will also take you a long time to generate reasonable
    # numbers of replicates for whatever analysis you are doing

# TLDR: You should look at print.runs = TRUE

#####################################################################

# Using the rate equation-input for complex diversification models

# First up... Diversity Dependent Models!
# Let's try Diversity-Dependent Branching over 50 time-units

# first, let's write the rate equation
# We'll use the diversity dependent rate equation model
    # from Ettienne et al. 2012 as an example here
    # Under this equation, p = q at carrying capacity K
# Many others are possible!
# Note that we don't need to use max(0,rate) as negative rates
    # are converted to zero by default, as controlled by
    # the argument negRatesAsZero

# From Ettiene et al.
    #   lambda = lambda0 - (lambda0 - mu)*(n/K)
# lambda and mu are branching rate and extinction rate
    # lambda and mu  ==  p and q in paleotree (i.e. Foote convention)
# lambda0 is the branching rate at richness = 0
# K is the carrying capacity
# n is the richness

# 'N' is the algebra symbol for standing taxonomic richness
    # for simFossilRecord's simulation capabilities
# also branching rate cannot reference extinction rate
# we'll have to set lambda0, mu and K in the rate equation directly

lambda0 <- 0.3  # branching rate at 0 richness in Ltu
K <- 40         # carrying capacity
mu <- 0.1       # extinction rate will 0.1 Ltu ( =  1/3 of lambda0 )

# technically, mu here represents the lambda at richness = K
    # i.e. lambdaK
# Ettienne et al. are just implicitly saying that the carrying capacity
    # is the richness at which lambda == mu
```

```
# construct the equation programmatically using paste0
branchingRateEq <- paste0(lambda0, "-(", lambda0, "-", mu, ")*(N/", K, ")")
# and take a look at it...
branchingRateEq
# its a thing of beauty, folks!

# now let's try it
records <- simFossilRecord(
    p = branchingRateEq,
    q = mu,
    nruns = 3,
    totalTime = 100,
    plot = TRUE,
    print.runs = TRUE
    )

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
    plotMultCurves = TRUE)

# those are some happy little diversity plateaus!


# now let's do diversity-dependent extinction

# let's slightly modify the model from Ettiene et al.
    #   mu = mu0 + (mu0 - muK)*(n/K)

mu0 <- 0.001     # mu at n = 0
muK <- 0.1       # mu at n = K (should be equal to lambda at K)
K <- 40          # carrying capacity (like above)
lambda <- muK    # equal to muK

# construct the equation programmatically using paste0
extRateEq <- paste0(mu0, "-(", mu0, "-", muK, ")*(N/" ,K, ")")
extRateEq

# now let's try it
records <- simFossilRecord(
    p = lambda,
    q = extRateEq,
    nruns = 3,
    totalTime = 100,
    plot = TRUE,
    print.runs = TRUE)

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
```

```
    plotMultCurves = TRUE)

# these plateaus looks a little more spiky
    #( maybe there is more turnover at K? )
# also, it took a longer for the rapid rise to occur

############################################################

# Now let's try an example with time-dependent origination
    # and extinction constrained to equal origination

# Note! Use of time-dependent parameters "D" and "T" may
# result in slower than normal simulation run times
# as the time-scale has to be discretized; see
# info for argument maxTimeStep above

# First, let's define a time-dependent rate equation
    # "T" is the symbol for time passed
timeEquation <- "0.4-(0.007*T)"

#in this equation, 0.4 is the rate at time = 0
    # and it will decrease by 0.007 with every time-unit
    # at time = 50, the final rate will be 0.05
# We can easily make it so extinction
    # is always equal to branching rate
# "P" is the algebraic equivalent for
     # "branching rate" in simFossilRecord

# now let's try it
records <- simFossilRecord(
    p = timeEquation,
    q = "P",
    nruns = 3,
    totalTime = 50,
    plot = TRUE,
    print.runs = TRUE
    )

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
    plotMultCurves = TRUE)

# high variability that seems to then smooth out as turnover decreases

# And duration what about duration-dependent processes?
    # let's do a duration-dep extinction equation:
durDepExt <- "0.01+(0.01*D)"

# okay, let's take it for a spin
records <- simFossilRecord(
    p = 0.1,
```

```
    q = durDepExt,
    nruns = 3,
    totalTime = 50,
    plot = TRUE,
    print.runs = TRUE
    )

records <- lapply(records,
    fossilRecord2fossilTaxa)

multiDiv(records,
    plotMultCurves = TRUE)

# creates runs full of short lived taxa

# Some more stuff to do with rate formulae!

# The formulae input method for rates allows
# for the rate to be a random variable

# For example, we could constantly redraw
  # the branching rate from an exponential

record <- simFossilRecord(
  p = "rexp(n = 1,rate = 10)",
  q = 0.1, r = 0.1, nruns = 1,
 nTotalTaxa = 50, plot = TRUE)

# Setting up specific time-variable rates can be laborious though
    # e.g. one rate during this 10 unit interval,
    # another during this interval, etc
# The problem is setting this up within a fixed function

############################################################
# Worked Example
# What if we want to draw a new rate from a
    # lognormal distribution every 10 time units?

# Need to randomly draw these rates *before* running simFossilTaxa
# This means also that we will need to individually do each simFossilTaxa run
    # since the rates are drawn outside of simFossilTaxa

# Get some reasonable log normal rates:
rates <- 0.1+rlnorm(100,meanlog = 1,sdlog = 1)/100

# Now paste it into a formulae that describes a function that
    # will change the rate output every 10 time units
rateEquation <- paste0(
                      "c(",
                      paste0(rates,collapse = ","),
                      ")[1+(T%/%10)]"
                      )
```

```
# and let's run it
record <- simFossilRecord(
    p = rateEquation,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    totalTime = c(30,40),
    plot = TRUE
    )

######################################################################

# Speciation Modes

# Some examples of varying the 'speciation modes' in simFossilRecord

# The default is pure budding cladogenesis
    # anag.rate = prop.bifurc = prop.cryptic = 0
# let's just set those for the moment anyway
record <- simFossilRecord(p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0, prop.bifurc = 0, prop.cryptic = 0,
    nruns = 1, nTotalTaxa = c(20,30) ,nExtant = 0, plot = TRUE)

#convert and plot phylogeny
    # note this will not reflect the 'budding' pattern
    # branching events will just appear like bifurcation
    # its a typical convention for phylogeny plotting
converted <- fossilRecord2fossilTaxa(record)
tree <- taxa2phylo(converted,plot = TRUE)

#now, an example of pure bifurcation
record <- simFossilRecord(p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0, prop.bifurc = 1, prop.cryptic = 0,
    nruns = 1, nTotalTaxa = c(20,30) ,nExtant = 0)
tree <- taxa2phylo(fossilRecord2fossilTaxa(record),plot = TRUE)

# all the short branches are due to ancestors that terminate
    # via pseudoextinction at bifurcation events

# an example with anagenesis = branching
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0.1,
    prop.bifurc = 0,
    prop.cryptic = 0,
    nruns = 1,
    nTotalTaxa = c(20,30),
    nExtant = 0
    )
tree <- taxa2phylo(fossilRecord2fossilTaxa(record),
    plot = TRUE)
# lots of pseudoextinction
```

```
# an example with anagenesis, pure bifurcation
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0.1,
    prop.bifurc = 1,
    prop.cryptic = 0,
    nruns = 1,
    nTotalTaxa = c(20,30) ,
    nExtant = 0
    )
tree <- taxa2phylo(
    fossilRecord2fossilTaxa(record),
    plot = TRUE
    )
# lots and lots of pseudoextinction

# an example with half cryptic speciation
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 0.5,
    nruns = 1,
    nTotalTaxa = c(20,30),
    nExtant = 0
    )

tree <- taxa2phylo(
    fossilRecord2fossilTaxa(record),
    plot = TRUE)

# notice that the tree has many more than the maximum of 30 tips:
    # that's because the cryptic taxa are not counted as
    # separate taxa by default, as controlled by count.cryptic

# an example with anagenesis, bifurcation, cryptic speciation
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0.1,
    prop.bifurc = 0.5,
    prop.cryptic = 0.5,
    nruns = 1,
    nTotalTaxa = c(20,30),
    nExtant = 0
    )

tree <- taxa2phylo(
    fossilRecord2fossilTaxa(record),
    plot = TRUE)

# note in this case, 50% of branching is cryptic
```

```
    # 25% is bifurcation, 25% is budding

# an example with anagenesis, pure cryptic speciation
    # morphotaxon identity will thus be entirely indep of branching!
    # I wonder if this is what is really going on, sometimes...
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0.1,
    prop.bifurc = 0,
    prop.cryptic = 1,
    nruns = 1,
    nTotalTaxa = c(20,30),
    nExtant = 0
    )
tree <- taxa2phylo(fossilRecord2fossilTaxa(record),
    plot = TRUE)

# merging cryptic taxa when all speciation is cryptic
set.seed(1)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    prop.crypt = 1,
    totalTime = 50,
    plot = TRUE
    )

# there looks like there is only a single taxon, but...
length(record)

#the above is the *actual* number of cryptic lineages

#############################################################################

# playing with count.cryptic with simulations of pure cryptic speciation
    # what if we had fossil records with NO morphological differentiation?

# We can choose to condition on total morphologically-distinguishable taxa
    # or total taxa including cryptic taxa with count.cryptic = FALSE

# an example with pure cryptic speciation with count.cryptic = TRUE
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 1,
    nruns = 1,
    totalTime = 50,
    nTotalTaxa = c(10,100),
    count.cryptic = TRUE
    )
```

```
tree <- taxa2phylo(fossilRecord2fossilTaxa(record))

# plot the tree
plot(tree)
axisPhylo()

# notice how the tip labels indicate all are the same morphotaxon?

#################
# an example with pure cryptic speciation with count.cryptic = FALSE
    # Need to be careful with this!

# We'll have to replace the # of taxa constraints with a time constraint
    # or else the count.cryptic = FALSE simulation will never end!

record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 1,
    nruns = 1,
    totalTime = 50,
    count.cryptic = FALSE
    )
tree <- taxa2phylo(fossilRecord2fossilTaxa(record))

# plot it
plot(tree)
axisPhylo()

########################################
# let's look at numbers of taxa returned when varying count.cryptic
    # with prop.cryptic = 0.5

# Count Cryptic Example Number One
# simple simulation going for 50 total taxa

# first, count.cryptic = FALSE (default)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 0.5,
    nruns = 1,
    nTotalTaxa = 50,
    count.cryptic = FALSE
    )

taxa <- fossilRecord2fossilTaxa(record)

#### Count the taxa/lineages !
```

```
# number of lineages (inc. cryptic)
nrow(taxa)

# number of morph-distinguishable taxa
length(unique(taxa[,6]))

###################

# Count Cryptic Example Number Two
# Now let's try with count.cryptic = TRUE

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 0.5,
    nruns = 1,
    nTotalTaxa = 50,
    count.cryptic = TRUE
    )

taxa <- fossilRecord2fossilTaxa(record)

### Count the taxa/lineages !
# number of lineages (inc. cryptic)
nrow(taxa)

# number of morph-distinguishable taxa
length(unique(taxa[,6]))
# okay...

############

# Count Cryptic Example Number Three
# now let's try cryptic speciation *with* 50 extant taxa!

# first, count.cryptic = FALSE (default)
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 0.5,
    nruns = 1,
    nExtant = 10,
    totalTime = c(1,100),
    count.cryptic = FALSE
    )

taxa <- fossilRecord2fossilTaxa(record)
```

```
### Count the taxa/lineages !
# number of still-living lineages (inc. cryptic)
sum(taxa[,5])

# number of still-living morph-dist. taxa
length(unique(taxa[taxa[,5] == 1,6]))

##############

# Count Cryptic Example Number Four
# like above with count.cryptic = TRUE

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    anag.rate = 0,
    prop.bifurc = 0,
    prop.cryptic = 0.5,
    nruns = 1,
    nExtant = 10,
    totalTime = c(1,100),
    count.cryptic = TRUE
    )
taxa <- fossilRecord2fossilTaxa(record)

### Count the taxa/lineages !
# number of still-living lineages (inc. cryptic)
sum(taxa[,5])
# number of still-living morph-dist. taxa
length(unique(taxa[taxa[,5] == 1,6]))

##################################################

# Specifying Number of Initial Taxa
    # Example using startTaxa to have more initial taxa

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    nTotalTaxa = 100,
    startTaxa = 20,
    plot = TRUE
    )

######################################################

# Specifying Combinations of Simulation Conditions

# Users can generate datasets that meet multiple conditions:
```

```
    # such as time, number of total taxa, extant taxa, sampled taxa
# These can be set as point conditions or ranges

# let's set time = 10-100 units, total taxa = 30-40, extant = 10
    #and look at acceptance rates with print.run
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    totalTime = c(10,100),
    nTotalTaxa = c(30,40),
    nExtant = 10,
    print.runs = TRUE,
    plot = TRUE
    )

# let's make the constraints on totaltaxa a little tighter
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    totalTime = c(50,100),
    nTotalTaxa = 30,
    nExtant = 10,
    print.runs = TRUE,
    plot = TRUE
    )

# still okay acceptance rates

# alright, now let's add a constraint on sampled taxa
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    totalTime = c(50,100),
    nTotalTaxa = 30,
    nExtant = 10,
    nSamp = 15,
    print.runs = TRUE,
    plot = TRUE
    )

# still okay acceptance rates

# we can be really odd and instead condition on having a single taxon
set.seed(1)

record <- simFossilRecord(
    p = 0.1,
```

```
    q = 0.1,
    r = 0.1,
    nTotalTaxa = 1,
    totalTime = c(10,20),
    plot = TRUE
    )

######################################################

# Simulations of Entirely Extinct Taxa

# Typically, a user may want to condition on a precise
    # number of sampled taxa in an all-extinct simulation

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 1,
    nTotalTaxa = c(1,100),
    nExtant = 0,
    nSamp = 20,
    print.runs = TRUE,
    plot = TRUE
    )

# Note that when simulations don't include
    # sampling or extant taxa, the plot
    # functionality changes
record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0,
    nruns = 1,
    nExtant = 0,
    print.runs = TRUE,
    plot = TRUE
    )

# Something similar happens when there is no sampling
    # and there are extant taxa but they aren't sampled

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0,
    nruns = 1,
    nExtant = 10,
    nTotalTaxa = 100,
    modern.samp.prob = 0,
    print.runs = TRUE,
    plot = TRUE
    )
```

```
#########################################################
# Retaining Rejected Simulations

# sometimes we might want to look at all the simulations
    # that don't meet acceptability criteria

# In particular, look at simulated clades that go extinct
    # rather than surviving long enough to satisfy
    # conditioning on temporal duration.

# Let's look for 10 simulations with following conditioning:
    # that are exactly 10 time-units in duration
    # that have between 10 and 30 total taxa
    # and have 1 to 30 extant taxa after 10 time-units

set.seed(4)

record <- simFossilRecord(
    p = 0.1,
    q = 0.1,
    r = 0.1,
    nruns = 10,
    totalTime = 10,
    nTotalTaxa = c(10,30),
    nExtant = c(1,30),
    returnAllRuns = TRUE,
    print.runs = TRUE,
    plot = TRUE
    )

# when returnAllRuns = TRUE, the length of record is 2
    # named 'accepted' and 'rejected'

# all the accepted runs (all 10) are in 'accepted'
length(record$accepted)

# all the rejected runs are in 'rejected'
length(record$rejected)

# probably many more than 10!
    # (I got 1770!)

# how many taxa are in each rejected simulation run?
totalTaxa_rej <- sapply(record$rejected, length)

# plot as a histogram
hist(totalTaxa_rej)
# a very nice exponential distribution...

# plot the rejected simulation with the most taxa

divCurveFossilRecordSim(
```

```
      fossilRecord = record$rejected[[
            which(max(totalTaxa_rej) == totalTaxa_rej)[1]
            ]]
      )

# we can plot all of these too...
result <- sapply(record$rejected,
     divCurveFossilRecordSim)

# let's look at the temporal duration of rejected clades

# need to write a function
getDuration <- function(record){
     taxa <- fossilRecord2fossilTaxa(record)
     maxAge <- max(taxa[,"orig.time"], na.rm = TRUE)
     minAge <- min(taxa[,"ext.time"], na.rm = TRUE)
     cladeDuration <- maxAge - minAge
     return(cladeDuration)
     }

# all the accepted simulations should have
       # identical durations (10 time-units)
sapply(record$accepted, getDuration)

# now the rejected set
durations_rej <- sapply(record$rejected, getDuration)
# plot as a histogram
hist(durations_rej)

# Most simulations hit the max time without
      # satisfying the other specified constraints
      # (probably they didn't have the min of 10 taxa total)
```

`simFossilRecordMethods` *Methods for Editing or Converting Output from Simulated Fossil Record Objects*

Description

These are a set of functions available for manipulating, translating and editing the objects of class `fossilRecordSimulation` output from function `simFossilRecord`.

Usage

```
timeSliceFossilRecord(
  fossilRecord,
  sliceTime,
```

```
  shiftRoot4TimeSlice = FALSE,
  modern.samp.prob = 1,
  tolerance = 10^-6
)

fossilRecord2fossilTaxa(fossilRecord)

fossilTaxa2fossilRecord(fossilTaxa)

fossilRecord2fossilRanges(
  fossilRecord,
  merge.cryptic = TRUE,
  ranges.only = TRUE
)
```

Arguments

`fossilRecord`	A list object output by `simFossilRecord`, often composed of multiple elements, each of which is data for 'one taxon', with the first element being a distinctive six-element vector composed of numbers, corresponding to the six fields in tables output by the deprecated function `simFossilTaxa`.
`sliceTime`	The date to slice the `simFossilRecord` output at, given in time-units before the modern, on the same scale as the input `fossilRecord`.
`shiftRoot4TimeSlice`	Should the dating of events be shifted, so that the date given for `sliceTime` is now 0, or should the dates not be shifted, so that they remain on the same scale as the input? This argument accepts a logical TRUE or FALSE, but also accepts the string `"withExtantOnly"`, which will only 'shift' the time-scale if living taxa are present, as determined by having ranges that overlap within `tolerance` of `sliceTime`.
`modern.samp.prob`	The probability that a taxon is sampled at the modern time (or, for `timeSliceFossilRecord`, the time at which the simulation data is slice). Must be a number between 0 and 1. If 1, all taxa that survive to the modern day (to the `sliceTime`) are sampled, if 0, none are.
`tolerance`	A small number which sets a range around the `sliceTime` within which taxa will be considered extant for the purposes of output.
`fossilTaxa`	A `fossilTaxa` object, composed of a table containing information on the true first and last appearance times of taxa, as well as their ancestor-descendant relationships.
`merge.cryptic`	If `TRUE`, sampling events for cryptic taxon-units (i.e. those in the same cryptic complex) will be merged into sampling events for a single taxon-unit (with the name of the first taxon in that cryptic complex).
`ranges.only`	If `TRUE` (the default), `fossilRecord2fossilRanges` will return the dates of the first and last sampled occurrences of each taxon-unit (i.e. the stratigraphic range of each taxon). If `FALSE`, instead a list will be output, with each element being a vector of dates for all sampling events of each taxon-unit.

Details

These functions exist to manipulate `fossilRecordSimulation` objects output from `simFossilRecord`, particularly so that they can be interfaced with functions in library `paleotree` in the same way that output from the deprecated 'legacy' simulation function `simFossilTaxa` was used.

`timeSliceFossilRecord` takes a given `fossilRecordSimulation` object and 'slices' the data to remove any events that occur after the given `sliceTime` and make it so any taxa still alive as of `sliceTime` are now listed as extant.

`fossilRecord2fossilTaxa` converts a `fossilRecordSimulation` object to the flat table format of taxon data as was originally output by deprecated function `simFossilTaxa`, and can be taken as input by a number of `paleotree` functions such as `sampleRanges`, `taxa2phylo` and `taxa2cladogram`.

`fossilTaxa2fossilRecord` does the reverse, converting a `simFossilTaxa` table into a `fossilRecordSimulation` list object, but returns a `fossilRecordSimulation` object that considers each species as *unsampled* (as sampling information is not contained within a `simFossilTaxa` table).

`fossilRecord2fossilRanges` converts a `fossilRecordSimulation` object to the flat table format of observed taxon ranges, as is typically output by processing `simFossilRecord` simulation output with `paleotree` function `sampleRanges`.

Value

Depends on the function and the arguments given. See Details.

Author(s)

David W. Bapst

See Also

`simFossilRecord`

Examples

```
set.seed(44)
record <- simFossilRecord(
    p = 0.1, q = 0.1, r = 0.1,
    nruns = 1,
    nTotalTaxa = c(20,30),
    nExtant = 0,
    plot = TRUE
    )

################################################
# time-slicing simulations at particular dates

# let's try slicing this record at 940 time-units
slicedRecord <- timeSliceFossilRecord(
    fossilRecord = record,
    sliceTime = 940
    )
```

```
# and let's plot it
divCurveFossilRecordSim(slicedRecord)

# now with shiftRoot4TimeSlice = TRUE to shift the root age
slicedRecord <- timeSliceFossilRecord(
    fossilRecord = record,
    sliceTime = 940,
    shiftRoot4TimeSlice = TRUE
    )
# and let's plot it
divCurveFossilRecordSim(slicedRecord)

# the last two plots look a little different
    # due to how axis limits are treated...
# notice that in both, 'modern' (extant) taxa
    # are sampled with probability = 1

########
# let's try it again, make that probability = 0
# now with shiftRoot4TimeSlice = TRUE

slicedRecord <- timeSliceFossilRecord(
    fossilRecord = record,
    sliceTime = 940,
    shiftRoot4TimeSlice = TRUE,
    modern.samp.prob = 0
    )

# and let's plot it
divCurveFossilRecordSim(slicedRecord)

############################

# converting to taxa objects and observed ranges

# convert to taxa data
taxa <- fossilRecord2fossilTaxa(record)
# convert to ranges
ranges <- fossilRecord2fossilRanges(record)

# plot diversity curves with multiDiv
multiDiv(list(taxa,ranges),
    plotMultCurves = TRUE)
# should look a lot like what we got earlier

# get the cladogram we'd obtain for these taxa with taxa2cladogram
cladogram <- taxa2cladogram(taxa,
    plot = TRUE)

# now get the time-scaled phylogenies with taxa2phylo

# first, with tips extending to the true times of extinction
treeExt <- taxa2phylo(taxa,
```

```
    plot = TRUE)

# now, with tips extending to the first appearance dates (FADs) of taxa
# get the FADs from the ranges
FADs <- ranges[,1]
treeFAD <- taxa2phylo(taxa,
    FADs,plot = TRUE)
```

SongZhangDicrano *Cladistic Data for Dicranograptid Graptolites from Song and Zhang (2014)*

Description

Character matrix and two cladograms for 13 dicranograptid (and outgroup) graptoloids, taken from Song and Zhang (2014). Included here for use with functions related to character change.

Format

Loading this dataset adds two objects to the R environment. `charMatDicrano` is a `data.frame` object composed of multiple factors, with `NA` values representing missing values (states coded as '?'), read in with `readNexus` from package `phylobase`. `cladogramDicranoX12` and `cladogramDicranoX13` are both cladograms, formatted as `phylo` class objects for use with package `ape`, without branch-lengths (as these was are, respectively, consensus tree and a maximum-parsimony tree from separate maximum-parsimony analyses).

Details

This example dataset is composed of a small cladistic character data for 13 taxa and 24 characters, taken from Song and Zhang (2014). Note that character 22 is a biostratigraphic character, which was not included in all analyses by Song and Zhang.

The first included cladogram `cladogramDicranoX12` is the majority-rule consensus of a maximum-parsimony analysis on 12 taxa (excluding on taxa with incompletely known anatomy) with 24 characters, including a biostratigraphic character. This tree is included here as, among the four trees depicted, it appeared to be the basis for the majority of Song and Zhang's discussion of dicranograptid systematics.

The second cladogram `cladogramDicranoX13` is a maximum-parsimony tree found by a maximum-parsimony analysis of 13 taxa with 24 characters, including a biostratigraphic character. This tree is much more resolved than the alternative majority-rule cladogram for 12 taxa.

The matrix and both trees were entered by hand from their flat graphic depiction in Song and Zhang's manuscript.

Source

Song, Y., and Y. Zhang. 2014. A preliminary study on the relationship of the early dicranograptids based on cladistic analysis. *GFF* 136(1):243-248.

Examples

```
data(SongZhangDicrano)

# Examining morphospace with a distance matrix

# calculate a distance matrix from the morph character data
char <- charMatDicrano[,-22] # remove strat character
charDist <- matrix(,nrow(char),nrow(char))
rownames(charDist) <- colnames(charDist) <- rownames(char)
for(i in 1:nrow(char)){for(j in 1:nrow(char)){
charDiff <- logical()
for(k in 1:ncol(char)){
selectPair <- char[c(i,j),k]
if(all(!is.na(selectPair))){
#drop states that are missing
isSame <- identical(selectPair[1],selectPair[2])
charDiff <- c(charDiff,isSame)
}
}
charDist[i,j] <- 1-sum(charDiff)/length(charDiff)
}}

#####
# PCO of character distance matrix

#can apply PCO (use lingoes correction to account for negative values
   #resulting from non-euclidean matrix
pco_res <- pcoa(charDist,correction = "lingoes")

#relative corrected eigenvalues
rel_corr_eig <- pco_res$values$Rel_corr_eig
layout(1:2)
plot(rel_corr_eig)
#cumulative
plot(cumsum(rel_corr_eig))

#well let's look at those PCO axes anyway
layout(1)
pco_axes <- pco_res$vectors
plot(pco_axes[,1],pco_axes[,2],pch = 16,
   xlab = paste("PCO Axis 1, Rel. Corr. Eigenvalue  = ",round(rel_corr_eig[1],3)),
   ylab = paste("PCO Axis 2, Rel. Corr. Eigenvalue  = ",round(rel_corr_eig[2],3)))

#######

# plot 12 taxon majority rule tree from Song and Zhang
plot(cladogramDicranoX12,
main = "MajRule_24charX12Taxa_wBiostratChar")

# plot 13 taxon MPT
plot(cladogramDicranoX13,
```

```
main = "MPT_24charX13Taxa_wBiostratChar")

##############

## Not run:
# Data was generated with following script:
require(ape)
require(phylobase)

charMatDicrano <- readNexus(file.choose(),type = "data",SYMBOLS = " 0 1 2")

cladogramDicranoX12 <- read.tree(file.choose())
cladogramDicranoX13 <- read.nexus(file.choose())

cladogramDicranoX13$tip.label <- rownames(
 charMatDicrano)[c(13,8,7,9,12,10,1,4,6,2,3,11,5)]

save(charMatDicrano,cladogramDicranoX12,file = "SongZhangDicrano.rdata")

## End(Not run)
```

`taxa2cladogram` *Convert Simulated Taxon Data into a Cladogram*

Description

Convert ancestor-descendant relationships of taxa into an 'ideal' unscaled cladogram, where taxa that could share true synapomorphies are arranged into nested clades.

Usage

```
taxa2cladogram(taxaData, drop.cryptic = FALSE, plot = FALSE)
```

Arguments

`taxaData`	A five-column matrix of taxonomic data, as output by `fossilRecord2fossilTaxa` via simulations produced using `simFossilRecord`. Previously, this was the default output of the deprecated function `simFossilTaxa`.
`drop.cryptic`	Should cryptic species be dropped (except for the first; effectively merging the cryptic species complexes into a single apparent species)? `drop.cryptic = FALSE` by default, so cryptic species are not dropped by default.
`plot`	If `TRUE` result the output with `ape::plot.phylo`.

Details

This function simulates an ideal cladistic process, where the relationships of a set of morphologically static taxa is resolved into a set of nested hierarchical relationships (a standard cladogram), as much as would be expected given the input relationships among those taxa. taxa2cladogram uses

information on the ancestor-descendant relationships of a bunch of taxa and constructs an unscaled cladogram of the hierarchically-nesting relationships among those taxa. There's no actual cladistics going on, this is just a simulation of that process. If there is any chance that a set of taxa could be resolved into a set of nested relationships given their ancestor-descendant relationships, they will be resolved so in the output of taxa2cladogram. No morphological characters are considered, we just assume that if there is a nesting relationship, then it could be resolved as such. This makes it the "ideal" cladogram of a simulated clade.

The result will probably not be fully resolved, as including both ancestor and descendant taxa will generally make it impossible to produce a fully nesting system of relationships. For example, consider a set of three morphologically-static taxa where the first is an ancestor (either direct or indirect, ala Foote, 1996) of both the second and third. If we imagine an ideal cladistic analysis of the morphological characters of those three taxa, this set of taxa will be unable to be broken up into bifurcating-nested relationships and thus result in a polytomy. Any set of ancestor-descendant relationships will have many of these, as some ancestors must have more than one descendant for the clade to diversify, as noted by Wagner and Erwin, 1995.

If there are cryptic taxa present in the output from `simFossilRecord`, these and any of their morphologically distinguishable descendants are collapsed into a polytomy to simulate the expected pattern of lack of phylogenetic resolution. In addition to this merging, cryptic taxa can be dropped via the argument drop.cryptic, such that only the first 'species' of each cryptic taxon assemblage is listed among the tip taxa (what we would actually expect to obtain, as we would not recognize cryptic taxa to be treated as different OTUs). By default, cryptic taxa are not dropped so that the same number of taxa as in the simulated data is retained.

Value

The resulting phylogeny without branch lengths is output as an object of class `phylo`.

The tip labels are the rownames from the simulation input; see documentation for `simFossilRecord` and `fossilRecord2fossilTaxa` documentation for details.

Author(s)

David W. Bapst

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141-151.

Wagner, P., and D. Erwin. 1995 Phylogenetic patterns as tests of speciation models. New approaches to speciation in the fossil record. Columbia University Press, New York:87-122.

See Also

`simFossilRecord`, `taxa2phylo`, fossilRecord2fossilTaxa

Examples

```
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
```

```
taxa <- fossilRecord2fossilTaxa(record)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
layout(1:2)
cladogram <- taxa2cladogram(taxa,plot = TRUE)
#compare the "real" time-scaled tree of taxon last occurrences (taxa2phylo)
     #to the 'ideal' cladogram
tree <- taxa2phylo(taxa,plot = TRUE)

#testing with cryptic speciation
recordCrypt <- simFossilRecord(p = 0.1, q = 0.1, prop.cryptic = 0.5, nruns = 1,
nTotalTaxa = c(30,40), nExtant = 0)
taxaCrypt <- fossilRecord2fossilTaxa(recordCrypt)
layout(1:2)
parOrig <- par(no.readonly = TRUE)
par(mar = c(0,0,0,0))
cladoCrypt1 <- taxa2cladogram(taxaCrypt,drop.cryptic = FALSE)
plot(cladoCrypt1)
cladoCrypt2 <- taxa2cladogram(taxaCrypt,drop.cryptic = TRUE)
plot(cladoCrypt2)

#reset plotting
par(parOrig)
layout(1)
```

`taxa2phylo` *Convert Simulated Taxon Data into a Phylogeny*

Description

Converts temporal and ancestor-descendant relationships of taxa into a dated phylogeny with tips at instantaneous points in time.

Usage

```
taxa2phylo(taxaData, obs_time = NULL, plot = FALSE)
```

Arguments

`taxaData`	A five-column matrix of taxonomic data, as output by `fossilRecord2fossilTaxa` via simulations produced using `simFossilRecord`. Previously, this was the default output of the deprecated function `simFossilTaxa`.
`obs_time`	A vector of per-taxon times of observation which must be in the same order of taxa as in the object `taxaData`. If `obs_time = NULL`, the LADs (column 4) in `taxaData` are used.
`plot`	If `TRUE` result the output with `ape::plot.phylo`.

Details

As described in the documentation for taxa2cladogram, the relationships among morphotaxa in the fossil record are difficult to describe in terms of traditional phylogenies. One possibility is to arbitrarily choose particular instantaneous points of time in the range of some taxa and describe the temporal relationships of the populations present at those dates. This is the tactic used by taxa2phylo.

By default, the dates selected (the obs_time argument) are the last occurrences of the taxon, so a simple use of this function will produce a dated tree which describes the relationships of the populations present at the last occurrence time of each taxon in the sampled data. Alternatively, obs_time can be supplied with different dates within the taxon ranges.

All data relating to when static morphotaxa appear or disappear in the record is lost. Branching points will be the actual time of speciation, which (under budding) will often be in the middle of the temporal range of a taxon.

Cryptic taxa are not dropped or merged as can be done with taxa2cladogram. The purpose of taxa2phylo is to obtain the 'true' pattern of evolution for the observation times, independent of what we might actually be able to recover, for the purpose of comparing in simulation analyses.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

The resulting phylogeny with branch lengths is output as an object of class phylo. This function will output trees with the element $root.time, which is the time of the root divergence in absolute time.

The tip labels are the row-names from the simulation input; see the documentation for simFossilRecord and fossilRecord2fossilTaxa for details.

Note

Do *NOT* use this function to date a real tree for a real dataset. It assumes you know the divergence/speciation times of the branching nodes and relationships perfectly, which is almost impossible given the undersampled nature of the fossil record. Use timePaleoPhy or cal3TimePaleoPhy instead.

Do use this function when doing simulations and you want to make a tree of the 'true' history, such as for simulating trait evolution along phylogenetic branches.

Unlike taxa2cladogram, this function does not merge cryptic taxa in output from simFossilRecord (via fossilRecord2fossilTaxa) and I do not offer an option to secondarily drop them. The tip labels should provide the necessary information for users to drop such taxa, however. See simFossilRecord.

Author(s)

David W. Bapst

See Also

simFossilRecord, taxa2cladogram, fossilRecord2fossilTaxa

Examples

```
set.seed(444)
record <- simFossilRecord(
   p = 0.1,
   q = 0.1,
   nruns = 1,
   nTotalTaxa = c(30,40),
   nExtant = 0
   )
taxa <- fossilRecord2fossilTaxa(record)
# let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
tree <- taxa2phylo(taxa)
phyloDiv(tree)

# now a phylogeny with tips placed at
   # the apparent time of extinction for each taxon
rangesCont <- sampleRanges(taxa,r = 0.5)
tree <- taxa2phylo(taxa,obs_time = rangesCont[,2])
phyloDiv(tree,drop.ZLB = FALSE)
#note that it drops taxa which were never sampled!

#testing with cryptic speciation
set.seed(444)
record <- simFossilRecord(
   p = 0.1,
   q = 0.1,
   prop.cryptic = 0.5,
   nruns = 1,
   nTotalTaxa = c(30,40),
   nExtant = 0,
   count.cryptic = TRUE
   )
taxaCrypt <- fossilRecord2fossilTaxa(record)
treeCrypt <- taxa2phylo(taxaCrypt)
layout(1)
plot(treeCrypt)
axisPhylo()
```

---

taxonSortPBDBocc *Sorting Unique Taxa of a Given Rank from Paleobiology Database Occurrence Data*

Description

Functions for sorting out unique taxa from Paleobiology Database occurrence downloads, which should accept several different formats resulting from different versions of the PBDB API and different vocabularies available from the API.

Usage

```
taxonSortPBDBocc(
  data,
  rank,
  onlyFormal = TRUE,
  cleanUncertain = TRUE,
  cleanResoValues = c(NA, "\"", "", "n. sp.", "n. gen.", " ", "  ")
)
```

Arguments

data	A table of occurrence data collected from the Paleobiology Database.
rank	The selected taxon rank; must be one of 'species', 'genus', 'family', 'order', 'class' or 'phylum'.
onlyFormal	If TRUE (the default) only taxa formally accepted by the Paleobiology Database are returned. If FALSE, then the identified name fields are searched for any additional 'informal' taxa with the proper taxon. If their taxon name happens to match any formal taxa, their occurrences are merged onto the formal taxa. This argument generally has any appreciable effect when rank = species.
cleanUncertain	If TRUE (the default) any occurrences with an entry in the respective 'resolution' field that is *not* found in the argument cleanResoValue will be removed from the dataset. These are assumed to be values indicating taxonomic uncertainty, i.e. 'cf.' or '?'.
cleanResoValues	The set of values that can be found in a 'resolution' field that do not cause a taxon to be removed, as they do not seem to indicate taxonomic uncertainty.

Details

Data input for taxonSortPBDBocc are expected to be from version 1.2 API with the 'pbdb' vocabulary. However, datasets are passed to internal function translatePBDBocc, which attempts to correct any necessary field names and field contents used by taxonSortPBDBocc.

This function can pull either *just* the 'formally' identified and synonymized taxa in a given table of occurrence data or pull *in addition* occurrences listed under informal taxa of the sought taxonomic rank. Only formal taxa are sorted by default; this is controlled by argument onlyFormal. Pulling the informally-listed taxonomic occurrences is often necessary in some groups that have received little focused taxonomic effort, such that many species are linked to their generic taxon ID and never received a species-level taxonomic ID in the PBDB. Pulling both formal and informally listed taxonomic occurrences is a hierarchical process and performed in stages: formal taxa are identified first, informal taxa are identified from the occurrences that are 'leftover', and informal occurrences with name labels that match a previously sorted formally listed taxon are concatenated to the 'formal' occurrences for that same taxon, rather than being listed under separate elements of the list as if they were separate taxa. This function is simpler than similar functions that inspired it by using the input"rank" to both filter occurrences and directly reference a taxon's accepted taxonomic placement, rather than a series of specific if() checks. Unlike some similar functions in other packages, such as version 0.3 paleobioDB's pbdb_temp_range, taxonSortPBDBocc does not check if sorted taxa have a single 'taxon_no' ID number. This makes the blanket assumption that if

a taxon's listed name in relevant fields is identical, the taxon is identical, with the important caveat that occurrences with accepted formal synonymies are sorted first based on their accepted names, followed by taxa without formal taxon IDs. This should avoid linking the same occurrences to multiple taxa by mistake, or assigning occurrences listed under separate formal taxa to the same taxon based on their 'identified' taxon name, as long as all formal taxa have unique names (note: this is an untested assumption). In some cases, this procedure is helpful, such as when taxa with identical generic and species names are listed under separate taxon ID numbers because of a difference in the listed subgenus for some occurrences (example, "Pseudoclimacograptus (Metaclimacograptus) hughesi' and 'Pseudoclimacograptus hughesi' in the PBDB as of 03/01/2015). Presumably any data that would be affected by differences in this procedure is very minor.

Occurrences with taxonomic uncertainty indicators in the listed identified taxon name are removed by default, as controlled by argument `cleanUncertain`. This is done by removing any occurrences that have an entry in `primary_reso` (was "`genus_reso`" in v1.1 API) when `rank` is a supraspecific level, and `species_reso` when `rank = species`, if that entry is not found in `cleanResoValues`. In some rare cases, when `onlyFormal = FALSE`, supraspecific taxon names may be returned in the output that have various 'cruft' attached, like 'n.sp.'.

Empty values in the input data table ("") are converted to NAs, as they may be due to issues with using read.csv to convert API-downloaded data.

Value

Returns a list where each element is different unique taxon obtained by the sorting function, and named with that taxon name. Each element is composed of a table containing all the same occurrence data fields as the input (potentially with some fields renamed and some field contents change, due to vocabulary translation).

Author(s)

David W. Bapst, but partly inspired by Matthew Clapham's `cleanTaxon` (found at this location on github) and R package `paleobioDB`'s `pbdb_temp_range` function (found at this location on github.

References

Peters, S. E., and M. McClennen. 2015. The Paleobiology Database application programming interface. *Paleobiology* 42(1):1-7.

See Also

Occurrence data as commonly used with `paleotree` functions can be obtained with `link{getPBDBocc}`. Occurrence data sorted by this function might be used with functions `occData2timeList` and `plotOccData`. Also, see the example graptolite dataset at `graptPBDB`

Examples

```
# getting occurrence data for a genus, sorting it
# Dicellograptus
dicelloData <- getPBDBocc("Dicellograptus")
```

```
dicelloOcc2 <- taxonSortPBDBocc(
   data = dicelloData,
   rank = "species",
   onlyFormal = FALSE
   )
names(dicelloOcc2)

# try a PBDB API download with lots of synonymization
#this should have only 1 species
# *old* way, using v1.1 of PBDB API:
# acoData <- read.csv(paste0(
# "http://paleobiodb.org/data1.1/occs/list.txt?",
# "base_name = Acosarina%20minuta&show=ident,phylo"))
#
# *new* method - with getPBDBocc, using v1.2 of PBDB API:
acoData <- getPBDBocc("Acosarina minuta")
acoOcc <- taxonSortPBDBocc(
   data = acoData,
   rank = "species",
   onlyFormal = FALSE
   )
names(acoOcc)



#load example graptolite PBDB occ dataset
data(graptPBDB)

#get formal genera
occGenus <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "genus"
   )
length(occGenus)

#get formal species
occSpeciesFormal <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "species")
length(occSpeciesFormal)

#yes, there are fewer 'formal'
   # graptolite species in the PBDB then genera

#get formal and informal species
occSpeciesInformal <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "species",
   onlyFormal = FALSE
   )
length(occSpeciesInformal)

#way more graptolite species are 'informal' in the PBDB
```

```
#get formal and informal species
#including from occurrences with uncertain taxonomy
#basically everything and the kitchen sink
occSpeciesEverything <- taxonSortPBDBocc(
   data = graptOccPBDB,
   rank = "species",
   onlyFormal = FALSE,
   cleanUncertain = FALSE)
length(occSpeciesEverything)
```

`taxonTable2taxonTree` *Create a Taxonomy-Based Phylogeny ('Taxon Tree') from a Hierarchical Table of Taxonomy Memberships*

Description

This function takes a matrix of taxon names, indicating a set of hierarchical taxonomic relationships conveyed as nested placements for a set of tip-taxa (listed in the last column of the matrix) and returns a 'taxonomy-tree' phylogeny object of class `phylo`.

Usage

```
taxonTable2taxonTree(taxonTable, cleanTree = TRUE, rootLabel = "root")
```

Arguments

`taxonTable`	A matrix of type character and multiple rows and columns, containing the tip taxa in the last column, one per row, with progressively larger taxa listed in prior columns (reading left-to-right). Invariant columns (i.e. taxa that all tip taxa are in) are allowed, but all but the most 'shallow' of such invariant taxa are dropped prior to transformation to a taxon-tree phylogeny object.
`cleanTree`	When `TRUE` (the default), the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If `FALSE`, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.
`rootLabel`	If the lowest constant/shared level in the taxonomic hierarchy isn't labeled, what label should be given to this level? The default is `"root"`.

Details

This function can deal with empty entries in cells of `taxonTable` by assuming these are lower-level taxa which are 'floating' freely somewhere in taxa several levels higher.

Value

A phylogeny of class phylo, where each tip is a taxon listed in the last column of the input taxonTable. Edges are scaled so that the distance from one taxon rank to another rank is one unit, then merged to remove singleton nodes. As not all taxa have parents at the immediate taxon level above, this leads to some odd cases. For example, two genera emanating from a node representing a class but with a very short (length = 1) branch and a long branch (length = 3) means one genus is simply placed in the class, with no family or order listed while the one on the long branch is within an order and family that is otherwise monogeneric.

The names of higher taxa than the tips should be appended as the element $node.label for the internal nodes.

Author(s)

David W. Bapst

See Also

makePBDBtaxonTree, parentChild2taxonTree

Examples

```
# let's create a small, really cheesy example
pokeTable <- rbind(cbind("Pokezooa","Shelloidea","Squirtadae",
     c("Squirtle","Blastoise","Wartortle")),
     c("Pokezooa","Shelloidea","","Lapras"),
     c("Pokezooa","","","Parasect"),
     cbind("Pokezooa","Hirsutamona","Rodentapokemorpha",
     c("Linoone","Sandshrew","Pikachu")),
     c("Pokezooa","Hirsutamona",NA,"Ursaring"))

pokeTree <- taxonTable2taxonTree(pokeTable)

plot(pokeTree)
nodelabels(pokeTree$node.label)
```

---

termTaxa *Simulating Extinct Clades of Monophyletic Taxa*

Description

This function simulates the diversification of clades composed of monophyletic terminal taxa, which are distinguished in a fashion completely alternative to way taxa are defined in the simulation functions simFossilRecord, taxa2cladogram and taxa2phylo.

Usage

```
simTermTaxa(ntaxa, sumRate = 0.2)

simTermTaxaAdvanced(
  p = 0.1,
  q = 0.1,
  mintaxa = 1,
  maxtaxa = 1000,
  mintime = 1,
  maxtime = 1000,
  minExtant = 0,
  maxExtant = NULL,
  min.cond = TRUE
)

trueTermTaxaTree(TermTaxaRes, time.obs)

deadTree(ntaxa, sumRate = 0.2)
```

Arguments

ntaxa	Number of monophyletic 'terminal' taxa (tip terminals) to be included on the simulated tree
sumRate	The sum of the instantaneous branching and extinction rates; see below.
p	Instantaneous rate of speciation/branching.
q	Instantaneous rate of extinction.
mintaxa	Minimum number of total taxa over the entire history of a clade necessary for a dataset to be accepted.
maxtaxa	Maximum number of total taxa over the entire history of a clade necessary for a dataset to be accepted.
mintime	Minimum time units to run any given simulation before stopping.
maxtime	Maximum time units to run any given simulation before stopping.
minExtant	Minimum number of living taxa allowed at end of simulations.
maxExtant	Maximum number of living taxa allowed at end of simulations.
min.cond	If TRUE, the default, simulations are stopped when they meet all minimum conditions. If FALSE, simulations will continue until they hit maximum conditions, but are only accepted as long as they still meet all minimum conditions in addition.
TermTaxaRes	The list output produced by simTermTaxa.
time.obs	A per-taxon vector of times of observation for the taxa in TermTaxaRes.

Details

deadTree generates a time-scaled topology for an entirely extinct clade of a specific number of tip taxa. Because the clade is extinct and assumed to have gone extinct in the distant past, many details

of typical birth-death simulators can be ignored. If a generated clade is already conditioned upon the (a) that some number of taxa was reached and (b) then the clade went extinct, the topology (i.e. the distribution of branching and extinction events) among the branches should be independent of the actual generating rate. The frequency of nodes is a simple mathematical function of the number of taxa (i.e. number of nodes is the number of taxa -1) and their placement should completely random, given that we generally treat birth-death processes as independent Poisson processes. Thus, in terms of generating the topology, this function is nothing but a simple wrapper for the ape function `rtree`, which randomly places splits among a set of taxa using a simple algorithm (see Paradis, 2012). To match the expectation of a birth-death process, new branch lengths are calculated as an exponential distribution with mean 1/`sumRate`, where `sumRate` represents the sum of the branching and extinction rates. Although as long as both the branching rate and extinction rates are more than zero, any non-ultrametric tree is possible, only when the two rates are non-zero and equal to each other will there be a high chance of getting an extinct clade with many tips. Any analyses one could do on a tree such as this will almost certainly give estimates of equal branching and extinction rates, just because all taxa are extinct.

`simTermTaxa` produces 'terminal-taxon' datasets; datasets of clades where the set of distinguishable taxa are defined as intrinsically monophyletic. (In version 1.6, I referred to this as the 'candle' mode, so named from the 'candling' horticultural practice and the visual conceptualization of the model.) On theoretical terms, terminal-taxa datasets are what would occur if (a) only descendant lineages can be sample and (b) all taxa are immediately differentiated as of the last speciation event and continue to be so differentiated until they go extinct. In practice, this means the taxa on such a tree would represent a sample of all the terminal branches, which start with some speciation event and end in an extinction event. These are taken to be the true original ranges of these taxa. No further taxa can be sampled than this set, whatsoever. Note that the differentiation here is a result of *a posteriori* consideration of the phylogeny: one can't even know what lineages could be sampled or the actual start points of such taxa until after the entire phylogeny of a group of organisms is generated.

Because all evolutionary history prior to any branching events is unsampled, this model is somewhat agnostic about the general model of differentiation among lineages. The only thing that can be said is that synapomorphies are assumed to be potentially present along every single branch, such that in an ideal scenario every clade could be defined. This would suggest very high anagenesis or bifurcation.

Because the set of observable taxa is a limited subset of the true evolution history, the true taxon ranges are not a faithful reproduction of the true diversity curve. See an example below.

`simTermTaxa` uses `deadTree` to make a phylogeny, so the only datasets produced are of extinct clades. `simTermTaxaAdvanced` is an alternative to `simTermTaxa` which uses `simFossilRecord` to generate the underlying pattern of evolutionary relationships and not `deadTree`. The arguments are thus similar to `simFossilRecord`, with some differences (as `simTermTaxaAdvanced` originally called the deprecated function `simFossilTaxa`). In particular, `simTermTaxaAdvanced` can be used to produce simulated datasets which have extant taxa.

`trueTermTaxaTree` is analogous to the function of `taxa2phylo`, in that it outputs the time-scaled-phylogeny for a terminal-taxon dataset for some times of observations. Unlike with the use of `taxa2phylo` on the output on `simFossilRecord` (via `fossilRecord2fossilTaxa`, there is no need to use `trueTermTaxaTree` to obtain the true phylogeny when times of extinction are the times of observation; just get the `$tree` element from the result output by `simTermTaxa`.

Also unlike with `taxa2phylo`, the cladistic topology of relationships among morphotaxa never changes as a function of time of observation. For obtaining the 'ideal cladogram' of relationships

among the terminal taxa, merely take the $tree element of the output from simtermTaxaData and remove the branch lengths (see below for an example).

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

deadTree gives a dated phylo object, with a $root.time element. As discussed above, the result is always an extinct phylogeny of exactly ntaxa.

simTermTaxa and simTermTaxaAdvanced both produce a list with two components: $taxonRanges which is a two-column matrix where each row gives the true first and last appearance of observable taxa and $tree which is a dated phylogeny with end-points at the true last appearance time of taxa.

trueTermTaxaTree produces a dated tree as a phylo object, which describes the relationships of populations at the times of observation given in the time.obs argument.

Author(s)

David W. Bapst

References

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

See Also

deadtree is simply a wrapper of the function rtree in ape.

For a very different way of simulating diversification in the fossil record, see simFossilRecord, fossilRecord2fossilTaxa, taxa2phylo and taxa2cladogram.

Examples

```
set.seed(444)
# example for 20 taxa
termTaxaRes <- simTermTaxa(20)

# let look at the taxa...
taxa <- termTaxaRes$taxonRanges
taxicDivCont(taxa)
# because ancestors don't even exist as taxa
# the true diversity curve can go to zero
# kinda bizarre!

# the tree should give a better idea
tree <- termTaxaRes$tree
phyloDiv(tree)
# well, okay, its a tree.

# get the 'ideal cladogram' ala taxa2cladogram
```

```
    # much easier with terminal-taxa simulations
    # as no paraphyletic taxa
cladogram <- tree
cladogram$edge.length <- NULL
plot(cladogram)

# trying out trueTermTaxaTree
# random times of observation: uniform distribution
time.obs <- apply(taxa,1,
    function(x) runif(1,x[2],x[1])
    )
tree1 <- trueTermTaxaTree(
    termTaxaRes,
    time.obs
    )
layout(1:2)
plot(tree)
plot(tree1)
layout(1)

##########################################
# let's look at the change in the terminal branches
plot(tree$edge.length,
    tree1$edge.length)
# can see some edges are shorter on the new tree, cool

# let's now simulate sampling and use FADs
layout(1:2)
plot(tree)
axisPhylo()

FADs <- sampleRanges(
    termTaxaRes$taxonRanges,
    r = 0.1)[,1]
tree1 <- trueTermTaxaTree(termTaxaRes, FADs)

plot(tree1)
axisPhylo()

##############################################
# can condition on sampling some average number of taxa
# analogous to deprecated function simFossilTaxa_SRcond
r <- 0.1
avgtaxa <- 50
sumRate <- 0.2

# avg number necc for an avg number sampled
ntaxa_orig <- avgtaxa / (r / (r + sumRate))
termTaxaRes <- simTermTaxa(
    ntaxa = ntaxa_orig,
    sumRate = sumRate)

# note that conditioning must be conducted using full sumRate
```

```
# this is because durations are functions of both rates
# just like in bifurcation

# now, use advanced version of simTermTaxa: simTermTaxaAdvanced
    # allows for extant taxa in a term-taxa simulation

#with min.cond
termTaxaRes <- simTermTaxaAdvanced(
    p = 0.1,
    q = 0.1,
    mintaxa = 50,
    maxtaxa = 100,
    maxtime = 100,
    minExtant = 10,
    maxExtant = 20,
    min.cond = TRUE
    )

# notice that arguments are similar to simFossilRecord
# and even more similar to deprecated function simFossilTaxa

plot(termTaxaRes$tree)
Ntip(termTaxaRes$tree)

# without min.cond
termTaxaRes <- simTermTaxaAdvanced(
    p = 0.1,
    q = 0.1,
    mintaxa = 50,
    maxtaxa = 100,
    maxtime = 100,
    minExtant = 10,
    maxExtant = 20,
    min.cond = FALSE
    )

plot(termTaxaRes$tree)
Ntip(termTaxaRes$tree)

layout(1)
```

`testEdgeMat` *Test the Edge Matrix of a '*`phylo`*' Phylogeny Object for Inconsistencies*

Description

`testEdgeMat` is a small simple function which tests the `$edge` matrix of `phylo` objects for inconsistencies that can cause downstream analytical problems. The associated function, `cleanNewPhylo` puts an input `phylo` object, presumably freshly created or reconstituted by some function, through a

series of post-processing, This includes having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions.

Usage

```
testEdgeMat(tree)

cleanNewPhylo(tree)
```

Arguments

tree	A phylogeny object of type phylo.

Details

Useful when doing complex manipulations of phylo objects (or reconstituting them, or their *de novo* construction), and thus is used by a number of paleotree functions.

Value

For testEdgeMat, if all the checks in the function pass correctly, the logical TRUE is returned.

For cleanNewPhylo, an object of class phylo is returned.

Author(s)

David W. Bapst, with a large number of tests incorporated from Emmanuel Paradis's checkValidPhylo function in package ape, (released under the GPL v>2).

Examples

```
set.seed(444)
tree <- rtree(10)
# should return TRUE
testEdgeMat(tree)

# should also work on star trees
testEdgeMat(stree(10))

# should also work on trees with two taxa
testEdgeMat(rtree(2))

# should also work on trees with one taxon
testEdgeMat(stree(1))

#running cleanNewPhylo on this tree should have little effect
#beyond ladderizing it...
tree1 <- cleanNewPhylo(tree)

#compare outputs
```

```
layout(1:2)
plot(tree)
plot(tree1)
layout(1)
```

`timeLadderTree` *Resolve Polytomies by Order of First Appearance*

Description

Resolves polytomies in trees with lineages arranged in a pectinate pattern (i.e. a ladder-like subtree), ordered by the time of first appearance (FAD) for each lineage.

Usage

```
timeLadderTree(tree, timeData)
```

Arguments

`tree`	A phylogeny, as an object of class `phylo`.
`timeData`	Two-column `matrix` of per-taxon first and last occurrences in absolute continuous time.

Details

This method of resolving polytomies assumes that the order of stratigraphic appearance perfectly depicts the order of branching. This may not be a good assumption for poorly sampled fossil records.

This function is for resolving trees when a continuous time-scale is known. For discrete time-scales, see the function `bin_timePaleoPhy`.

Taxa with the same identical first appearance date will be ordered randomly. Thus, the output is slightly stochastic, but only when ties exist. This is probably uncommon with real data on continuous time-scales.

Taxa not shared between the input tree and the `timeData` matrix, or listed as having a FAD or LAD of `NA` in `timeData` will be dropped and will not be included in the output tree.

See this blog post for more information:

http://nemagraptus.blogspot.com/2012/07/resolving-polytomies-according-to.html

Value

Returns the modified tree as an object of class `phylo`, with no edge lengths.

Author(s)

David W. Bapst

See Also

di2multi

Examples

```
set.seed(444)
record <- simFossilRecord(p = 0.1, q = 0.1, nruns = 1,
nTotalTaxa = c(100,200))
taxa <- fossilRecord2fossilTaxa(record)
tree <- taxa2cladogram(taxa)
ranges <- sampleRanges(taxa,r = 0.5)
tree1 <- timeLadderTree(tree,ranges)
layout(1:2)
plot(ladderize(tree),show.tip.label = FALSE)
plot(ladderize(tree1),show.tip.label = FALSE)

#an example with applying timeLadderTree to discrete time data
rangeData <- binTimeData(ranges,int.len = 5) #sim discrete range data
tree2 <- bin_timePaleoPhy(tree,timeList = rangeData,timeres = TRUE)
plot(ladderize(tree),show.tip.label = FALSE)
plot(ladderize(tree2),show.tip.label = FALSE)
axisPhylo()

layout(1)
```

---

timeList2fourDate *Converting Datasets of Taxon Ranges in Intervals Between* timeList *format and* fourDate *format*

Description

Functions for manipulating data where the first and last appearances of taxa are known from bounded intervals of time. The two main functions listed here are for converting between (1) a data structure consisting of a single 'flat' table where each taxon is listed as a set of four dates (a fourDate data type), and (2) a list format where each taxon is listed as its first and last intervals, with an associated table of age bounds for the intervals referred to in the first table (referred to as a timeList data structure by many paleotree functions).

Usage

```
timeList2fourDate(timeList)

fourDate2timeList(fourDate)
```

Arguments

timeList	A list composed of two matrices with two columns each, the first giving interval start and end date bounds, and the second giving taxon first and last interval appearances in reference to the intervals listed in the first matrix.
fourDate	A four column matrix where each row is a different taxon, the first two columns are the lower and upper bounds on the time of first appearance for that taxon and the third and fourth columns are respectively the lower and upper bounds on the time of last appearance for that taxon, all in time before present.

Details

timeList2fourDate is for converting from a timeList format to a fourDate format. fourDate2timeList is for converting from a fourDate format to a timeList format.

Value

A converted data object, respective to the function applied.

Author(s)

David W. Bapst

References

See my recent blog post on temporal datasets in paleontology for some details:

http://nemagraptus.blogspot.com/2015/02/how-do-we-treat-fossil-age-data-dates.html

See Also

bin_timePaleoPhy and taxicDivDisc for common applications; binTimeData for a simulation function for such data objects

Examples

```
# timeList object from the retiolinae dataset
data(retiolitinae)

str(retioRanges)

taxicDivDisc(retioRanges)

fourDateRet <- timeList2fourDate(retioRanges)

# total uncertainty in retio first and last appearances?
sum(
    (fourDateRet[,1] - fourDateRet[,2]) +
    (fourDateRet[,3]-fourDateRet[,4])
    )

#convert back
```

```
newTimeList <- fourDate2timeList(fourDateRet)
taxicDivDisc(retioRanges)
```

`timePaleoPhy` *Simplistic* a posteriori *Dating Approaches For Paleontological Phylogenies*

Description

Dates an unscaled cladogram of fossil taxa using information on their temporal ranges, using various methods. Also can resolve polytomies randomly and output samples of randomly-resolved trees. As simple methods of dating ('time-scaling') phylogenies of fossil taxa can have biasing effects on macroevolutionary analyses (Bapst, 2014, Paleobiology), this function is largely retained for legacy purposes and plotting applications. The methods implemented by the functions listed here do **not** return realistic estimates of divergence dates, and users are strongly encouraged to investigate other methods such as `cal3TimePaleoPhy` or `createMrBayesTipDatingNexus`.

Usage

```
timePaleoPhy(
  tree,
  timeData,
  type = "basic",
  vartime = NULL,
  ntrees = 1,
  randres = FALSE,
  timeres = FALSE,
  add.term = FALSE,
  inc.term.adj = FALSE,
  dateTreatment = "firstLast",
  node.mins = NULL,
  noisyDrop = TRUE,
  plot = FALSE
)

bin_timePaleoPhy(
  tree,
  timeList,
  type = "basic",
  vartime = NULL,
  ntrees = 1,
  nonstoch.bin = FALSE,
  randres = FALSE,
  timeres = FALSE,
  sites = NULL,
  point.occur = FALSE,
```

```
  add.term = FALSE,
  inc.term.adj = FALSE,
  dateTreatment = "firstLast",
  node.mins = NULL,
  noisyDrop = TRUE,
  plot = FALSE
)
```

Arguments

tree	An unscaled cladogram of fossil taxa, of class phylo. Tip labels must match the taxon labels in the respective temporal data.
timeData	Two-column matrix of first and last occurrences in absolute continuous time, with row names as the taxon IDs used on the tree. This means the first column is very precise FADs (first appearance dates) and the second column is very precise LADs (last appearance dates), reflect the precise points in time when taxa first and last appear. If there is stratigraphic uncertainty in when taxa appear in the fossil record, it is preferable to use the bin_ dating functions; however, see the argument dateTreatment.
type	Type of time-scaling method used. Can be "basic", "equal", "equal_paleotree_legacy", "equal_date.phylo_legacy" "aba", "zbla" or "mbl". Type = "basic" by default. See details below.
vartime	Time variable; usage depends on the type argument. Ignored if type = "basic".
ntrees	Number of dated trees to output. Only applicable is there is some stochastic (random) element to the analysis. If ntrees is greater than one, and both randres = FALSE and dateTreatment is neither 'minMax' or 'randObs', the function will fail and a warning is issued, as these arguments would simply produce multiple identical time-scaled trees.
randres	Should polytomies be randomly resolved? By default, timePaleoPhy does not resolve polytomies, instead outputting a dated tree that is only as resolved as the input tree. If randres = TRUE, then polytomies will be randomly resolved using multi2di from the package ape. If randres = TRUE and ntrees = 1, a warning is printed that users should analyze multiple randomly-resolved trees, rather than a single such tree, although a tree is still output.
timeres	Should polytomies be resolved relative to the order of appearance of lineages? By default, timePaleoPhy does not resolve polytomies, instead outputting a time-scaled tree that is only as resolved as the input tree. If timeres = TRUE, then polytomies will be resolved with respect to time using the paleotree function timeLadderTree. See that functions help page for more information; the result of time-order resolving of polytomies generally does not differ across multiple uses, unlike use of multi2di.
add.term	If TRUE, adds terminal ranges. By default, this function will not add the ranges of taxa when time-scaling a tree, so that the tips correspond temporally to the first appearance datums of the given taxa. If add.term = TRUE, then the 'terminal ranges' of the taxa are added to the tips after tree is dated, such that the tips now correspond to the last appearance datums.

inc.term.adj If TRUE, includes terminal ranges in branch length estimates for the various adjustment of branch lengths under all methods except type = 'basic' (in other words, a terminal length branch will not be treated as zero length if inc.term.adj = TRUE, if the tip-taxon on this branch has a non-zero duration). By default, this argument is FALSE and this function will not include the ranges of taxa when adjusting branch lengths, so that zero-length branches before first appearance times will be extended. An error is returned if this inc.term.adj = TRUE but type = "basic" or add.term = FALSE, as this argument is inconsistent with those argument options.

dateTreatment This argument controls the interpretation of timeData. The default setting dateTreatment = "firstLast" treats the dates in timeData as a column of precise first and last appearances.

A second option is dateTreatment = "minMax", which treats these dates as minimum and maximum bounds on single point dates. Under this option, all taxa in the analysis will be treated as being point dates, such that the first appearance is also the last. These dates will be pulled under a uniform distribution. If dateTreatment = "minMax" is used, add.term becomes meaningless, and the use of it will return an error message.

A third option is dateTreatment = "randObs". This assumes that the dates in the matrix are first and last appearance times, but that the desired time of observation is unknown. Thus, this is much like dateTreatment = "firstLast" except the effective time of observation (the taxon's LAD under dateTreatment = "firstLast") is treated as an uncertain date, and is randomly sampled between the first and last appearance times. The FAD still is treated as a fixed number, used for dating the nodes. In previous versions of paleotree, this was called in timePaleoPhy using the argument rand.obs, which has been removed for clarity. This temporal uncertainty in times of observation might be useful if a user is interested in applying phylogeny-based approaches to studying trait evolution, but have per-taxon measurements of traits that come from museum specimens with uncertain temporal placement.

With both arguments dateTreatment = "minMax" and dateTreatment = "randObs", the sampling of dates from random distributions should compel users to produce many time-scaled trees for any given analytical purpose. Note that dateTreatment = "minMax" returns an error in 'bin' time-scaling functions; please use points.occur instead.

node.mins The minimum dates of internal nodes (clades) on a phylogeny can be set using node.mins. This argument takes a vector of the same length as the number of nodes, with dates given in the same order as nodes are ordered in the tree$edge matrix. Note that in tree$edge, terminal tips are given the first set of numbers (1:Ntip(tree)), so the first element of node.mins is the first internal node (the node numbered Ntip(tree)+1, which is generally the root for most phylo objects read by read.tree). Not all nodes need be given minimum dates; those without minimum dates can be given as NA in node.mins, but the vector must be the same length as the number of internal nodes in tree. These are minimum date constraints, such that a node will be forced to be *at least as old as this date*, but the final date may be even older depending on the taxon dates used, the time-scaling method applied, the vartime used and any other minimum node dates given (e.g. if a clade is given a very old minimum date, this will (of course)

over-ride any minimum dates given for clades that that node is nested within). Although `vartime` does adjust the node age downwards when the equal method is used, if a user has a specific date they'd like to constrain the root to, they should use `node.mins` instead because the result is more predictable.

`noisyDrop` If `TRUE` (the default), any taxa dropped from tree due to not having a matching entry in the time data will be listed in a system message.

`plot` If `TRUE`, plots the input and output phylogenies.

`timeList` A list composed of two matrices giving interval times and taxon appearance dates. The rownames of the second matrix should be the taxon IDs, identical to the `tip.labels` for tree. See details.

`nonstoch.bin` If `nonstoch.bin = TRUE` (the default is `FALSE`, dates are *not* stochastically drawn from uniform distributions bounded by the upper and lower boundaries of the geologic intervals (the 'bins'), as typically occurs with '`bin_`' time-scaling methods in `paleotree` but instead first-appearance dates are assigned to the earliest time of the interval a taxon first appears in, while last-appearance dates are placed at the youngest (the 'later-most') date in the interval that that taxon last appears in. This option may be useful for plotting. Note that if `nonstoch.bin = TRUE`, the `sites` argument becomes arbitrary and has no influence on the output.

`sites` Optional two column matrix, composed of site IDs for taxon FADs and LADs. The sites argument allows users to constrain the placement of dates by restricting multiple fossil taxa whose FADs or LADs are from the same very temporally restricted sites (such as fossil-rich Lagerstatten) to always have the same date, across many iterations of time-scaled trees. To do this, provide a `matrix` to the `sites` argument where the "site" of each FAD and LAD for every taxon is listed, as corresponding to the second matrix in `timeList`. If no sites matrix is given (the default), then it is assumed all fossil come from different "sites" and there is no shared temporal structure among the events.

`point.occur` If true, will automatically produce a 'sites' matrix which forces all FADs and LADs to equal each other. This should be used when all taxa are only known from single 'point occurrences', i.e. each is only recovered from a single bed/horizon, such as a Lagerstatten.

Details

Simplistic a posteriori *Dating ('Time-Scaling') Methods for Paleontology Phylogenies*

These functions are an attempt to unify and collect previously used and discussed *a posteriori* methods for time-scaling phylogenies of fossil taxa. Unfortunately, it can be difficult to attribute some time-scaling methods to specific references in the literature.

There are five main *a posteriori* approaches that can be used by `timePaleoPhy`. Four of these main types use some value of absolute time, chosen *a priori*, to date the tree. This is handled by the argument `vartime`, which is `NULL` by default and unused for type `"basic"`.

"basic" This most simple of time-scaling methods ignores `vartime` and scales nodes so they are as old as the first appearance of their oldest descendant (Smith, 1994). This method produces many zero-length branches (Hunt and Carrano, 2010).

"equal" The "equal" method defined by G. Lloyd and used in Brusatte et al. (2008) and Lloyd et al. (2012). Originally usable in code supplied by G. Lloyd, the "equal" algorithm is recreated here as closely as possible. This method works by increasing the time of the root divergence by some amount and then adjusting zero-length branches so that time on early branches is re-apportioned out along those later branches equally. Branches are adjusted in order relative to the number of nodes separating the edge from the root, going from the furthest (most shallow) edges to the deepest edges. The choice of ordering algorithm can have an unanticipated large effect on the resulting dated trees created using "equal" and it appears that paleotree and functions written by G. Lloyd were not always consistent. The default option described here was only introduced in paleotree and other available software sources in August 2014. Thus, two legacy "equal" methods are included in this function, so users can emulate older ordering algorithms for "equal" which are now deprecated, as they do not match the underlying logic of the original "equal" algorithm and do not minimize down-passes when adjusting branch lengths on the time-scaled tree.

The root age can be adjusted backwards in time by either increasing by an arbitrary amount (via the vartime argument) or by setting the root age directly (via the node.mins argument); conversely, the function will also allow a user to opt to not alter the root age at all.

"equal_paleotree_legacy" Exactly like "equal" above, except that edges are ordered instead by their depth (i.e. number of nodes from the root). This minor modified version was referred to as "equal" for this timePaleoPhy function in paleotree until February 2014, and thus is included here solely for legacy purposes. This ordering algorithm does not minimize branch adjustment cycles, like the newer default offered under currently "equal".

"equal_date.phylo_legacy" Exactly like "equal" above, except that edges are ordered relative to their time (i.e., the total edge length) from the root following the application of the 'basic' time-scaling method, exactly as in G. Lloyd's original application. This was the method for sorting edges in the "equal" algorithm in G. Lloyd's date.phylo script and DatePhylo in package strap until August 2014, and was the default "equal" algorithm in paleotree's timePaleoPhy function from February 2014 until August 2014. This ordering algorithm does not minimize branch adjustment cycles, like the newer default offered under currently "equal". Due to how the presence of zero-length branches can make ordering branches based on time to be very unpredictable, this version of the "equal" algorithm is **highly not recommended**.

"aba" All branches additive. This method takes the "basic" time-scaled tree and adds vartime to all branches. Note that this time-scaling method can (and often will) warp the tree structure, leading to tips to originate out of order with the appearance data used.

"zlba" Zero-length branches additive. This method adds vartime to all zero-length branches in the "basic" tree. Discussed (possibly?) by Hunt and Carrano, 2010. Note that this time-scaling method can warp the tree structure, leading to tips to originate out of order with the appearance data used.

"mbl" Minimum branch length. Scales all branches so they are greater than or equal to vartime, and subtract time added to later branches from earlier branches in order to maintain the temporal structure of events. A version of this was first introduced by Laurin (2004).

These functions cannot time-scale branches relative to reconstructed character changes along branches, as used by Lloyd et al. (2012). Please see DatePhylo in R package strap for this functionality.

These functions will intuitively drop taxa from the tree with NA for range or are missing from timeData or timeList. Taxa dropped from the tree will be will be listed in a message output to the user. The same is done for taxa in the timeList object not listed in the tree.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

As of August 2014, please note that the branch-ordering algorithm used in "equal" has changed to match the current algorithm used by DatePhylo in package strap, and that two legacy versions of "equal" have been added to this function, respectively representing how timePaleoPhy and DatePhylo (and its predecessor date.phylo) applied the "equal" time-scaling method.

Interpretation of Taxon Ages in timePaleoPhy

timePaleoPhy is *primarily* designed for direct application to datasets where taxon first and last appearances are precisely known in continuous time, with no stratigraphic uncertainty. This is an uncommon form of data to have from the fossil record, although not an impossible form (micropaleontologists often have very precise range charts, for example). Instead, most data has some form of stratigraphic uncertainty. However, for some groups, the more typical 'first' and 'last' dates found in the literature or in databases represent the minimum and maximum absolute ages for the fossil collections that a taxon is known is known from. Presumably, the first and last appearances of that taxon in the fossil record is at unknown dates within these bounds.

As of paleotree v2.0. the treatment of taxon ages in timePaleoPhy is handled by the argument dateTreatment. *By default,* this argument is set to "firstLast" which means the matrix of ages are treated as precise first and last appearance dates (i.e. FADs and LADs). The earlier FADs will be used to calibrate the node ages, which could produce fairly nonsensical results if these are 'minimum' ages instead and reflect age uncertainty. Alternatively, dateTreatment can be set to "minMax" which instead treats taxon age data as minimum and maximum bounds on a single point date. These point dates, if the minimum and maximum bounds option is selected, are chose under a uniform distribution. Many dated trees should be generated, in order to approximate the uncertainty in the dates. Additionally, there is a third option for dateTreatment: users may also make it so that the 'times of observation' of trees are uncertain, such that the tips of the tree (with terminal ranges added) should be randomly selected from a uniform distribution. Essentially, this third option treats the dates as first and last appearances, but treats the first appearance dates as known and fixed, but the 'last appearance' dates as unknown. In previous versions of paleotree, this third option was enacted with the argument rand.obs, which has been removed for clarity.

Interpretation of Taxon Ages in bin_timePaleoPhy

As an alternative to using timePaleoPhy, bin_timePaleoPhy is a wrapper of timePaleoPhy which produces time-scaled trees for datasets which only have interval data available. For each output tree, taxon first and last appearance dates are placed within their listed intervals under a uniform distribution. Thus, a large sample of dated trees will (hopefully) approximate the uncertainty in the actual timing of the FADs and LADs. In some ways, treating taxonomic age uncertainty may be more logical via bin_timePaleoPhy, as it is tied to specific interval bounds, and there are more options available for certain types of age uncertainty, such as for cases where specimens come from the same fossil site.

The input timeList object for bin_timePaleoPhy can have overlapping (i.e. non-sequential) intervals, and intervals of uneven size. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. If taxa occur only in single collections (i.e. their first and last appearance in the fossil record is synchronous, the argument point.occur will force all taxa to have instantaneous durations in the fossil record. Otherwise, by default, taxa are assumed to first and last appear in the fossil record at different points in time, with some positive duration. The sites matrix can be used to force only a portion of taxa to have simultaneous first and last appearances.

If timeData or the elements of timeList are actually data.frames (as output by read.csv or read.table), these will be coerced to a matrix.

Tutorial

A tutorial for applying the time-scaling functions in paleotree, along with an example using real (graptolite) data, can be found here: http://nemagraptus.blogspot.com/2013/06/a-tutorial-to-cal3-time-scalin html

Value

The output of these functions is a time-scaled tree or set of time-scaled trees, of either class phylo or multiphylo, depending on the argument ntrees. All trees are output with an element $root.time. This is the time of the root on the tree and is important for comparing patterns across trees. Note that the $root.time element is defined relative to the earliest first appearance date, and thus later tips may seem to occur in the distant future under the "aba" and "zbla" time-scaling methods.

Trees created with bin_timePaleoPhy will output with some additional elements, in particular $ranges.used, a matrix which records the continuous-time ranges generated for time-scaling each tree. (Essentially a pseudo-timeData matrix.)

Note

Please account for stratigraphic uncertainty in your analysis. Unless you have exceptionally resolved data, select an appropriate option in dateTreatment within timePaleoPhy, use the more sophisticated bin_timePaleoPhy or code your own wrapper function of timePaleoPhy that accounts for stratigraphic uncertainty in your dataset.

Author(s)

David W. Bapst, heavily inspired by code supplied by Graeme Lloyd and Gene Hunt.

References

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Bapst, D. W. 2014. Assessing the effect of time-scaling methods on phylogeny-based analyses in the fossil record. **Paleobiology 40**(3):331-351.

Brusatte, S. L., M. J. Benton, M. Ruta, and G. T. Lloyd. 2008 Superiority, Competition, and Opportunism in the Evolutionary Radiation of Dinosaurs. *Science* **321**(5895):1485-91488.

Hunt, G., and M. T. Carrano. 2010 Models and methods for analyzing phenotypic evolution in lineages and clades. In J. Alroy, and G. Hunt, eds. Short Course on Quantitative Methods in Paleobiology. Paleontological Society.

Laurin, M. 2004. The Evolution of Body Size, Cope's Rule and the Origin of Amniotes. *Systematic Biology* 53(4):594-622.

Lloyd, G. T., S. C. Wang, and S. L. Brusatte. 2012 Identifying Heterogeneity in Rates of Morphological Evolution: Discrete Character Change in the Evolution of Lungfish(Sarcopterygii, Dipnoi). *Evolution* **66**(2):330–348.

Smith, A. B. 1994 Systematics and the fossil record: documenting evolutionary patterns. Blackwell Scientific, Oxford.

See Also

cal3TimePaleoPhy, binTimeData, multi2di

For an alternative time-scaling function, which includes the 'ruta' method that weights the time-scaling of branches by estimates of character change along with implementations of the 'basic' and "equal" methods described here, please see function DatePhylo in package strap.

Examples

```
# examples with empirical data

#load data
data(retiolitinae)

#Can plot the unscaled cladogram
plot(retioTree)
#Can plot discrete time interval diversity curve with retioRanges
taxicDivDisc(retioRanges)

#Use basic time-scaling (terminal branches only go to FADs)
ttree <- bin_timePaleoPhy(
    tree = retioTree,
    timeList = retioRanges,
    type = "basic",
    ntrees = 1,
    plot = TRUE
    )

#Use basic time-scaling (terminal branches go to LADs)
ttree <- bin_timePaleoPhy(
    tree = retioTree,
    timeList = retioRanges,
    type = "basic",
    add.term = TRUE,
    ntrees = 1,
    plot = TRUE
    )

#mininum branch length time-scaling (terminal branches only go to FADs)
ttree <- bin_timePaleoPhy(
    tree = retioTree,
    timeList = retioRanges,
    type = "mbl",
    vartime = 1,
    ntrees = 1,
    plot = TRUE
    )

####################

# examples with simulated data
```

```
# Simulate some fossil ranges with simFossilRecord
set.seed(444)
record <- simFossilRecord(
    p = 0.1, q = 0.1,
    nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = 0
    )
taxa <- fossilRecord2fossilTaxa(record)

#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa, r = 0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,
    plot = TRUE)

#Now let's try timePaleoPhy using the continuous range data
ttree <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    plot = TRUE
    )

#plot diversity curve
phyloDiv(ttree)


################################################
# that tree lacked the terminal parts of ranges
    # (tips stops at the taxon FADs)
# let's add those terminal ranges back on with add.term
ttree <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    add.term = TRUE,
    plot = TRUE
    )

#plot diversity curve
phyloDiv(ttree)


################################################
# that tree didn't look very resolved, does it?
   # (See Wagner and Erwin 1995 to see why)
# can randomly resolve trees using the argument randres
# each resulting tree will have polytomies
   # randomly resolved stochastically using ape::multi2di
ttree <- timePaleoPhy(
    cladogram,
```

```
    rangesCont,
    type = "basic",
    ntrees = 1,
    randres = TRUE,
    add.term = TRUE,
    plot = TRUE
    )

# Notice the warning it prints! PAY ATTENTION!
# We would need to set ntrees to a large number
    # to get a fair sample of trees

# if we set ntrees > 1, timePaleoPhy will make multiple time-trees
ttrees <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    ntrees = 9,
    randres = TRUE,
    add.term = TRUE,
    plot = TRUE)
#let's compare nine of them at once in a plot
layout(matrix(1:9, 3, 3))
parOrig <- par(no.readonly = TRUE)
par(mar = c(1, 1, 1, 1))
for(i in 1:9){
    plot(
        ladderize(ttrees[[i]]),
        show.tip.label = FALSE,
        no.margin = TRUE
        )
    }
#they are all a bit different!


##############################################
# we can also resolve the polytomies in the tree
    # according to time of first appearance via the function timeLadderTree
    # by setting the argument 'timeres = TRUE'
ttree <- timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    ntrees = 1,
    timeres = TRUE,
    add.term = TRUE,
    plot = TRUE
    )

#can plot the median diversity curve with multiDiv
layout(1)
par(parOrig)
multiDiv(ttrees)
```

```
#compare different methods of timePaleoPhy
layout(matrix(1:6, 3, 2))
parOrig <- par(no.readonly = TRUE)
par(mar = c(3, 2, 1, 2))
plot(ladderize(timePaleoPhy(
    cladogram,
    rangesCont,
    type = "basic",
    vartime = NULL,
    add.term = TRUE
    )))
axisPhylo()
text(x = 50,y = 23,"type = basic",adj = c(0,0.5),cex = 1.2)
#
plot(ladderize(timePaleoPhy(
    cladogram,
    rangesCont,
    type = "equal",
    vartime = 10,
    add.term = TRUE
    )))
axisPhylo()
text(x = 55,y = 23,"type = equal",adj = c(0,0.5),cex = 1.2)
#
plot(ladderize(timePaleoPhy(
    cladogram,
    rangesCont,
    type = "aba",
    vartime = 1,
    add.term = TRUE
    )))
axisPhylo()
text(x = 55,y = 23,"type = aba",adj = c(0,0.5),cex = 1.2)
#
plot(ladderize(timePaleoPhy(
    cladogram,
    rangesCont,
    type = "zlba",
    vartime = 1,
    add.term = TRUE
    )))
axisPhylo()
text(x = 55, y = 23, "type = zlba",
     adj = c(0,0.5), cex = 1.2)
#
plot(ladderize(timePaleoPhy(
    cladogram,
    rangesCont,
    type = "mbl",
    vartime = 1,
    add.term = TRUE
    )))
```

```
axisPhylo()
text(x = 55,y = 23,"type = mbl",adj = c(0,0.5),cex = 1.2)
layout(1)
par(parOrig)


##############################################
#using node.mins
#let's say we have (molecular??) evidence that
    # node #5 is at least 1200 time-units ago
#to use node.mins, first need to drop any unshared taxa

droppers <- cladogram$tip.label[is.na(
    match(cladogram$tip.label,
         names(which(!is.na(rangesCont[,1])))
         )
    )]
cladoDrop <- drop.tip(cladogram, droppers)

# now make vector same length as number of nodes
nodeDates <- rep(NA, Nnode(cladoDrop))
nodeDates[5] <- 1200
ttree1 <- timePaleoPhy(
    cladoDrop,rangesCont,
    type = "basic",
    randres = FALSE,
    node.mins = nodeDates,
    plot = TRUE)
ttree2 <- timePaleoPhy(
    cladoDrop,
    rangesCont,
    type = "basic",
    randres = TRUE,
    node.mins = nodeDates,
    plot = TRUE)


##################################################
##################################################
##################################################
#Using bin_timePaleoPhy to time-scale with discrete interval data

#first let's use binTimeData() to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length = 1)

ttreeB1 <- bin_timePaleoPhy(
    cladogram,
    rangesDisc,
    type = "basic",
    ntrees = 1,
    randres = TRUE,
    add.term = TRUE,
    plot = FALSE
```

```
    )

#notice the warning it prints!
phyloDiv(ttreeB1)

#with time-order resolving via timeLadderTree
ttreeB2 <- bin_timePaleoPhy(
    cladogram,
    rangesDisc,
    type = "basic",
    ntrees = 1,
    timeres = TRUE,
    add.term = TRUE,
    plot = FALSE
    )

phyloDiv(ttreeB2)


#can also force the appearance timings not to be chosen stochastically
ttreeB3 <- bin_timePaleoPhy(
    cladogram,rangesDisc,
    type = "basic",
    ntrees = 1,
    nonstoch.bin = TRUE,
    randres = TRUE,
    add.term = TRUE,
    plot = FALSE
    )

phyloDiv(ttreeB3)


# testing node.mins in bin_timePaleoPhy
ttree <- bin_timePaleoPhy(
    cladoDrop,
    rangesDisc,
    type = "basic",
    ntrees = 1,
    add.term = TRUE,
    randres = FALSE,
    node.mins = nodeDates,
    plot = TRUE
    )

# with randres = TRUE
ttree <- bin_timePaleoPhy(
    cladoDrop,
    rangesDisc,
    type = "basic",ntrees = 1,
    add.term = TRUE,
    randres = TRUE,
    node.mins = nodeDates,
```

```
    plot = TRUE
    )


#simple three taxon example for testing inc.term.adj
ranges1 <- cbind(c(3,4,5), c(2,3,1))
rownames(ranges1) <- paste("t", 1:3, sep = "")
clado1 <- read.tree(file = NA,
    text = "(t1,(t2,t3));")
ttree1 <- timePaleoPhy(
    clado1,
    ranges1,
    type = "mbl",
    vartime = 1
    )
ttree2 <- timePaleoPhy(
    clado1,
    ranges1,
    type = "mbl",
    vartime = 1,
    add.term = TRUE
    )
ttree3 <- timePaleoPhy(
    clado1,
    ranges1,
    type = "mbl",
    vartime = 1,
    add.term = TRUE,
    inc.term.adj = TRUE
    )

# see differences in root times
ttree1$root.time
ttree2$root.time
ttree3$root.time
-apply(ranges1,1,diff)

layout(1:3)
plot(ttree1)
axisPhylo()
plot(ttree2)
axisPhylo()
plot(ttree3)
axisPhylo()
```

`timeSliceTree` *Time-Slicing a Phylogeny*

Description

Removes the portion of a tree after a set point in time, as if the tree after that moment had been sliced away.

Usage

```
timeSliceTree(
  ttree,
  sliceTime,
  drop.extinct = FALSE,
  tipLabels = "earliestDesc",
  plot = TRUE
)
```

Arguments

ttree	A time-scaled phylogeny of class phylo.
sliceTime	Time at which to 'slice' the tree. See details.
drop.extinct	If TRUE, drops tips that go extinct before the input timeSlice using function dropExtinct. Note that dropExtinct will also automatically adjust the $root.time if the removal of extinct branches causes the room to shift to a younger age.
tipLabels	What sort of tip labels should be placed on cropped branches which had multiple descendants? The default option, "earliestDesc" labels a clipped branch with the earliest appearing tip descendant of that branch. Alternatively, if tipLabels = "allDesc", these tips can instead be labeled with a compound label consisting of all descendants that were on the cropped branch, separated by semi-colons.
plot	If TRUE, plots input and output trees for comparison.

Details

The function assumes that the input ttree will generally have an element called $root.time, which is the time before present that the root divergence occurred. If $root.time is not present as an element of ttree, then it is assumed the tip furthest from the root is at time zero (present-day) and a new $root.time is calculated (a warning will be issued in this case).

The sliceTime is always calculated as on the same scale as ttree$root.time. In other words, if root.time = 100, then timeSlice = 80 will slice the tree 20 time units after the root.

If drop.extinct = TRUE, then extinct tips are dropped and (if present) the $root.time of ttree is adjusted. This is done using the paleotree function dropExtinct.

Value

Returns the modified phylogeny as an object of class phylo. See argument tipLabels for how the labeling of the tips for cut branches is controlled.

Note

Note that the default behavior of `tiplabels = "earliestDesc"` labels cut branches with the tip label for the earliest tip descendant. This is somewhat arbitrary; the actual morphotaxon present at that time might have been a different taxon that the earliest appearing tip. For simulated datasets where morphotaxon identity is known throughout and not limited to tip observations, slice the taxon data in that more detailed form, and then transform that morphotaxon data to a tree, perhaps with `taxa2phylo`.

Author(s)

David W. Bapst, with modification of code by Klaus Schliep to avoid use of function `dist.nodes`, which has difficulty with large trees, and greatly benefiting the run time of this function.

See Also

`phyloDiv`, `dropExtinct`, `dropExtant`

Also see the function `treeSlice` in the library `phytools`, which will slice a tree at some point in and return all the subtrees which remain after the slicing time. (Effectively the reversed *opposite* of `timeSliceTree`.)

Examples

```
# a neat example of using phyloDiv with timeSliceTree
    # to simulate doing extant-only phylogeny studies
    # of diversification...in the past!
set.seed(444)
record <- simFossilRecord(
    p = 0.1, q = 0.1, nruns = 1,
    nTotalTaxa = c(30,40),
    nExtant = 0)
taxa <- fossilRecord2fossilTaxa(record)
taxicDivCont(taxa)

# that's the whole diversity curve
   # now let's do it for a particular time-slide
tree <- taxa2phylo(taxa)
# use timeSliceTree to make tree of relationships
    # up until time = 950
tree950 <- timeSliceTree(
    tree,
    sliceTime = 950,
    plot = TRUE,
    drop.extinct = FALSE
    )

# compare tip labels when we use tipLabels = "allDesc"
tree950_AD <- timeSliceTree(
    tree,
    sliceTime = 950,
    plot = TRUE,
```

```
    tipLabel = "allDesc",
    drop.extinct = FALSE
    )

# look for the differences!
cbind(tree950$tip.label, tree950_AD$tip.label)

# with timeSliceTree we could
    # look at the lineage accumulation curve
    # we would recover from the species extant
    # at that point in time

# use drop.extinct = T to only get the
    # tree of lineages extant at time = 950
tree950 <- timeSliceTree(
    tree,
    sliceTime = 950,
    plot = FALSE,
    drop.extinct = TRUE
    )
# now its an ultrametric tree with many fewer tips...
    # lets plot the lineage accumulation plot on a log scale
phyloDiv(tree950,
    plotLogRich = TRUE
    )
```

`tipDatingCompatabilitySummaryMrB` *Get the Compatibility Summary Topology From a Tip-Dating Analysis with MrBayes*

Description

This function is designed to avoid methodological issues with getting sensible consensus summary topologies from posteriors samples of tip-dated, sampled-ancestor trees output by Mr Bayes. This function will obtain samples of posterior trees, from external files, remove the specified burn-in, and output an undated summary tree of clades (splits) indicated on the output tree, as a particular posterior probability threshold. Posterior probabilities may be appended to the nodes of the output phylogeny. This function should be used for examining topological variation in the posterior.

Usage

```
tipDatingCompatabilitySummaryMrB(
  runFile,
  nRuns = 2,
  burnin = 0.5,
  compatibilityThreshold = 0.5,
  labelPostProb = TRUE
)
```

Arguments

runFile	A filename in the current directory, or a path to a file that is either a *.p* or *.t* file from a MrBayes analysis. This filename and path will be used for finding additional *.t* and *.p* files, via the nRuns settings and assuming that files are in the same directory *and* these files are named under typical MrBayes file naming conventions. (In other words, if you have renamed your *.p* or *.t* files, this function probably won't be able to find them.)
nRuns	The number of runs in your analysis. This variable is used for figuring out what filenames will be searched for: if you specify that you have less runs than you actually ran in reality, then some runs won't be examined in this function. Conversely, specify too many, and this function will throw an error when it cannot find files it expects but do not exist. The default for this argument (*two* runs) is based on the default number of runs in MrBayes.
burnin	The fraction of trees sampled in the posterior discarded and not returned by this function directly, nor included in calculation of summary trees. Must be a numeric value greater than 0 and less than 1.
compatibilityThreshold	The posterior probability threshold (between 1 and zero, post-burn-in) that a node must satisfy to appear on the output summary tree. The default is 0.5, making the trees output half-compatibility trees (summary topologies), similar to the majority-rule consensus commonly used in maximum parsimony analyses. The value cannot be lower than 0.5 due to current technical constraints, and the need for an R function that iteratively ranks possible splits to be included in a consensus, as the consensus is calculated. Currently, if a clade frequency threshold given (argument p) to ape function consensus, which is used internally by halfCompatTree, all nodes above that compatibility threshold, even splits which are contradictory, will be included on the output tree, often resulting in uninterpretable output.
labelPostProb	Logical. If TRUE, then nodes of the output tree will be labeled with their respective posterior probabilities, as calculated based on the frequency of a clade occurring across the post-burn-in posterior tree sample. If FALSE, this is skipped.

Details

This function is most useful for dealing with dating analyses in MrBayes, particularly when tip-dating a tree with fossil taxa, as the half-compatibility and all-compatibility summary trees offered by the 'sumt' command in MrBayes can have issues properly portraying summary trees from such datasets.

Summary topologies calculated with some tip-dating software environments, such as MrBayes, can be subject to strange and uninterpretable methodological artifacts as the methods use attempt to present summary topologies with branch lengths. Many of these algorithms as currently implemented cannot handle the two-degree nodes or zero-length branches that arise from having sampled ancestors. Users looking to summarize a tip-dating analysis cannot easily calculate a dated summary: if they want a dated tree, they *must* examine a single tree from the posterior (either randomly selected or chosen based on some criteria such as marginal likelihood, posterior probability, etc). However, if our main interest is the unscaled evolutionary closeness of taxonomic units without

reference to time, then it is sufficient to examine a summary of the topological variation over our posterior.

Value

A single, undated summary tree, containing those clades (splits) found in greater frequency in the post-burn-in posterior tree sample more than the value of `compatibilityThreshold`, of class `phylo`. If `labelPostProb = TRUE`, nodes will be labeled with the posterior probability of the respective clade.

Note

Consensus trees that combine clades found different trees in the same tree sample may inadvertently combine clades that are not found on any of the actual trees sampled in the posterior, and may be quite far from the posterior trees as sampled in multivariate tree-space. This is a standard criticism leveled at consensus-type summary trees, except for the strict consensus (equivalent here to if a user tried `compatibilityThreshold = 1`). However, post-burn-in posterior tree samples often sample (and thus contain) a considerable range of tree-space within them, and thus the strict consensus (a total compatibility tree?)

Author(s)

David W. Bapst

See Also

See function `obtainDatedPosteriorTreesMrB` for additional ways of processing and evaluating trees from MrBayes posterior samples.

Summary trees are estimated using the function `consensus` in package ape.

Examples

```
## Not run:
#pull post-burn-in trees from the posterior
      # and get the half-compatibility summary (majority-rule consensus)
      # by setting 'compatibilityThreshold = 0.5'

halfCompatTree <- tipDatingCompatabilitySummaryMrB(
  runFile = "C:\\myTipDatingAnalysis\\MrB_run_fossil_05-10-17.nex.run1.t",
  nRuns = 2, burnin = 0.5,
  compatibilityThreshold = 0.5,
  labelPostProb = TRUE
  )

# let's try plotting it with posterior probabilities as node labels
plot(halfCompatTree)
nodelabels(halfCompatTree$node.label)


## End(Not run)
```

`treeContradiction` *Measure the Contradiction Difference Between Two Phylogenetic Topologies*

Description

An alternative measure of pair-wise dissimilarity between two tree topologies which ignores differences in phylogenetic resolution between the two, unlike typical metrics (such as Robinson-Foulds distance). The metric essentially counts up the number of splits on both trees that are directly contradicted by a split on the contrasting topology (treating both as unrooted). By default, this 'contradiction difference' value is then scaled to between 0 and 1, by dividing by the total number of splits that could have been contradicted across both trees (2 * (Number of shared tips - 2)). On this scaled, 0 represents no conflicting relationships and 1 reflects two entirely conflicting topologies, similar to the rescaling in Colless's consensus fork index.

Usage

```
treeContradiction(tree1, tree2, rescale = TRUE)
```

Arguments

`tree1, tree2`	Two phylogenies, with the same number of tips and an identical set of tip labels, both of class `phylo`.
`rescale`	A logical. If `FALSE`, the raw number of contradicted splits across both trees is reported. If `TRUE` (the default), the contradiction difference value is returned rescaled to the total number of splits across both input trees that could have contradicted.

Details

Algorithmically, conflicting splits are identified by counting the number of splits (via ape's `prop.part`) on one tree that disagree with at least one split on the other tree: for example, split (AB)CD would be contradicted by split (AC)BD. To put it another way, all we need to test for is whether the taxa segregated by that split were found to be more closely related to some other taxa, not so segregated by the considered split.

This metric was designed mainly for use with trees that differ in their resolution, particularly when it is necessary to compare between summary trees (such as consensus trees of half-compatibility summaries) from separate phylogenetic analyses. Note that comparing summary trees can be problematic in some instances, and users should carefully consider their question of interest, and whether it may be more ideal to consider whole samples of trees (e.g., the posterior sample, or the sample of most parsimonious trees).

The contradiction difference is *not* a metric distance: most notably, the triangle inequality is not held and thus the 'space' it describes between topologies is not a metric space. This can be shown most simply when considering any two different but fully-resolve topologies and a third topology that is a star tree. The star tree will have a zero pair-wise CD with either fully-resolved phylogeny,

but there will be a positive CD between the fully-resolved trees. An example of this is shown in the examples below.

The CD also suggest very large differences when small numbers of taxa shift greatly across the tree, a property shared by many other typical tree comparisons, such as RF distances. See examples below.

Value

The contradiction difference between two trees is reported as a single numeric variable.

Author(s)

David W. Bapst. This code was produced as part of a project funded by National Science Foundation grant EAR-1147537 to S. J. Carlson.

References

This contradiction difference measure was introduced in:

Bapst, D. W., H. A. Schreiber, and S. J. Carlson. 2018. Combined Analysis of Extant Rhynchonellida (Brachiopoda) using Morphological and Molecular Data. *Systematic Biology* 67(1):32-48.

See Also

See phangorn's function for calculating the Robinson-Foulds distance: `treedist`.

Graeme Lloyd's `metatree` package, currently not on CRAN, also contains the function `MultiTreeDistance` for calculating both the contradiction difference measure and the Robinson-Foulds distance. This function is optimized for very large samples of trees or very large trees, and thus may be faster than `treeContradiction`. Also see the function `MultiTreeContradiction` in the same package.

Examples

```
# let's simulate two trees

set.seed(1)
treeA <- rtree(30,br = NULL)
treeB <- rtree(30,br = NULL)

## Not run:

# visualize the difference between these two trees
library(phytools)
plot(cophylo(treeA,treeB))

# what is the Robinson-Foulds (RF) distance between these trees?
library(phangorn)
treedist(treeA,treeB)


## End(Not run)
```

```
# The RF distance is less intuitive when
    # we consider a tree that isn't well-resolved

# let's simulate the worst resolved tree possible: a star tree
treeC <- stree(30)

## Not run:
# plot the tanglegram between A and C
plot(cophylo(treeA,treeC))

# however the RF distance is *not* zero
# even though the only difference is a difference in resolution
treedist(treeA,treeC)

## End(Not run)

# the contradiction difference (CD) ignores differences in resolution

# Tree C (the star tree) has zero CD between it and trees A and B
identical(treeContradiction(treeA,treeC),0)  # should be zero distance
identical(treeContradiction(treeB,treeC),0)  # should be zero distance

# two identical trees also have zero CD between them (as you'd hope)
identical(treeContradiction(treeA,treeA),0)  # should be zero distance

#' and here's the CD between A and B
treeContradiction(treeA,treeB)  # should be non-zero distance

# a less ideal property of the CD is that two taxon on opposite ends of the
# moving from side of the topology to the other of an otherwise identical tree
# will return the maximum contradiction difference possible (i.e., ` =  1`)

# an example
treeAA <- read.tree(text = "(A,(B,(C,(D,(E,F)))));")
treeBB <- read.tree(text = "(E,(B,(C,(D,(A,F)))));")

## Not run:
plot(cophylo(treeAA,treeBB))

## End(Not run)

treeContradiction(treeAA,treeBB)

## Not run:
# Note however also a property of RF distance too:
treedist(treeAA,treeBB)

## End(Not run)
```

`twoWayEcologyCluster` *R-Mode vs Q-Mode Two-Way Cluster Analyses and Abundance Plot for Community Ecology Data*

Description

This mode plots both R-mode (across sites) and Q-mode (across taxa) dendrograms for a community ecology dataset, with branches aligned with a grid of dots representing the relative abundance of taxa at each site in the dataset.

Usage

```
twoWayEcologyCluster(
  xDist,
  yDist,
  propAbund,
  clustMethod = "average",
  marginBetween = 0.1,
  abundExpansion = 3,
  cex.axisLabels = 1,
  xAxisLabel = "Across Sites",
  yAxisLabel = "Across Taxa"
)
```

Arguments

`xDist`	The pair-wise distance matrix for the cluster diagram drawn along the horizontal axis of the graphic. Should be a distance matrix, or a matrix that can be coerced to a distance matrix, for the same number of units as rows in `propAbund`.
`yDist`	The pair-wise distance matrix for the cluster diagram drawn along the vertical axis of the graphic. Should be a distance matrix, or a matrix that can be coerced to a distance matrix, for the same number of units as columns in `propAbund`.
`propAbund`	A matrix of abundance data, preferably relative abundance scaled as proportions of the total number of individuals at each site. This data determines the size scale of the taxon/site dots.
`clustMethod`	The agglomerative clustering method used, as with argument `method` with function `hclust`. `clustMethod` must be one of `"average"` (the default method for this function, also known as average-linkage or as UPGMA), `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"mcquitty"` (also known as WPGMA), `"median"` (also known as WPGMC) or `"centroid"` (also known as UPGMC).
`marginBetween`	Argument controlling space placed between the cluster diagrams and the abundance plot. Default is 0.1.
`abundExpansion`	An argument that is a multiplier controlling the size of dots plotted for reflecting relative abundance.
`cex.axisLabels`	Character expansion parameter for controlling the plotting of axis labels on the abundance dot-grid only.

xAxisLabel	The label placed on the horizontal axis of the plot.
yAxisLabel	The label placed on the vertical axis of the plot.

Details

You might be able to apply this to datasets that aren't community ecology datasets of proportional abundance, but I can't guarantee or even predict what will happen.

Value

This function creates a plot, and returns nothing, not even invisible output.

Author(s)

David W. Bapst

References

The function here was designed to emulate previous published 'two-way' cluster diagrams, particularly the one in Miller, 1988:

Miller, A. I. 1988. Spatial Resolution in Subfossil Molluscan Remains: Implications for Paleobiological Analyses. *Paleobiology* 14(1):91-103.

See Also

Several other functions for community ecology data in paleotree are described at the communityEcology help file. Also see the example dataset, kanto.

Examples

```
set.seed(1)

# generate random community ecology data
# using a Poisson distribution
data<-matrix(rpois(5*7,1),5,7)

# get relative abundance, distance matrices
propAbundMat<-t(apply(data,1,function(x) x/sum(x)))
rownames(propAbundMat)<-paste0("site ", 1:nrow(propAbundMat))
colnames(propAbundMat)<-paste0("taxon ", 1:ncol(propAbundMat))

# for simplicity, let's calculate
# the pairwise square chord distance
# between sites and taxa

squareChordDist<-function(mat){
res<-apply(mat,1,function(x)
apply(mat,1,function(y)
sum((sqrt(x)-sqrt(y))^2)
)
)
```

```
#
res<-as.dist(res)
return(res)
}

# its not a very popular distance metric
# but it will do
# quite popular in palynology

siteDist<-squareChordDist(propAbundMat)
taxaDist<-squareChordDist(t(propAbundMat))

dev.new(width=10)

twoWayEcologyCluster(
xDist = siteDist,
yDist = taxaDist,
propAbund = propAbundMat
)

## Not run:

# now let's try an example with the example kanto dataset
# and use bray-curtis distance from vegan

library(vegan)

data(kanto)

# get distance matrices for sites and taxa
# based on bray-curtis dist
# standardized to total abundance

# standardize site matrix to relative abundance
siteStandKanto <- decostand(kanto, method = "total")

# calculate site distance matrix (Bray-Curtis)
siteDistKanto <- vegdist(siteStandKanto, "bray")

# calculate taxa distance matrix (Bray-Curtis)
# from transposed standardized site matrix
taxaDistKanto <- vegdist(t(siteStandKanto), "bray")

dev.new(width=10)

twoWayEcologyCluster(
    xDist = siteDistKanto,
    yDist = taxaDistKanto,
    propAbund = siteStandKanto,
    cex.axisLabels = 0.8
    )
```

```
## End(Not run)
```

`unitLengthTree` *Scale Tree to Unit-Length*

Description

Rescales all edges of a phylogeny to be equal to a single unit (1, or "unit-length").

Usage

```
unitLengthTree(tree)
```

Arguments

`tree`	A phylogeny as an object of class `phylo`.

Details

Probably not a good way to scale a tree for comparative studies. What does it mean to scale every edge of the phylogeny to the same length?

This is not a rhetorical question. First, consider that on a 'reconstructed' tree with only extant taxa, it would mean assuming the time between births of new lineages that survive to the modern is extremely constant over evolutionary history (because the unit-length wouldn't change, unlike the birth-death model, which assumes lineages that survive to the modern accumulate at an accelerating exponential rate, even with constant birth and death rates).

A paleontological tree (say, under the Fossilized Birth-Death Model) treated with this 'unit-length' approach would assuming constancy and rigid homogeneity of the timing between the birth (origination events) of new lineages that (a) survive to the modern day, or (b) are sampled at some future point in the fossil record. We should assume even with constant extinction and fossilization rates that such lineages should occur more frequently as we approach the present-day.

Note that in neither of those cases, the 'unit-length' branch-scaling approach does not produce trees whose edge lengths somehow represent the 'speciational' model, where evolutionary change is entirely 'cladogenetic' (ala punctuated equilibrium) and associated only with branching events. This would only be true on the true, perfectly sampled tree, which isn't what anyone has.

Thus, overall, the value of the 'unit-length' approach is rather questionable.

Value

Returns the modified phylogeny as an object of class `phylo`. Any `$root.time` element is removed.

See Also

As an alternative to using `unitLengthTree` in comparative studies, see `timePaleoPhy`. Or nearly anything, really...

See also `speciationalTree` in the package geiger, which does essentially the same thing as `unitLengthTree`.

Examples

```
set.seed(444)
tree <- rtree(10)

layout(1:2)
plot(tree)
plot(unitLengthTree(tree))
layout(1)
```

Index