Package 'cladoRcpp'

July 22, 2025

Type Package

Title C++ Implementations of Phylogenetic Cladogenesis Calculations

Version 0.15.1

Date 2018-11-21

Author Nicholas J. Matzke [aut, cre, cph]

Maintainer Nicholas J. Matzke <nickmatzke.ncse@gmail.com>

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp

Description Various cladogenesis-related calculations that are slow in pure R are implemented in C++ with Rcpp. These include the calculation of the probability of various scenarios for the inheritance of geographic range at the divergence events on a phylogenetic tree, and other calculations necessary for models which are not continuous-time markov chains (CTMC), but where change instead occurs instantaneously at speciation events. Typically these models must assess the probability of every possible combination of (ancestor state, left descendent state, right descendent state). This means that there are up to (# of states)^3 combinations to investigate, and in biogeographical models, there can easily be hundreds of states, so calculation time becomes an issue. C++ implementation plus clever tricks (many combinations can be eliminated a priori) can greatly speed the computation time over naive R implementations. CITATION INFO: This package is the result of my Ph.D. research, please cite the package if you use it! Type: citation(package=``cladoRcpp") to get the citation information.

URL http://phylo.wikidot.com/biogeobears

License GPL (>= 2)

LazyLoad yes

ByteCompile true

RoxygenNote 6.1.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-11-21 05:30:02 UTC

Contents

`cladoRcpp-package` *Phylogenetic probability calculations using Rcpp*

Description

Cladogenic probability calculations using Rcpp

Details

Package:	cladoRcpp
Type:	Package
Version:	0.15.1
Date:	2018-11-21
License:	GPL (>= 3)
LazyLoad:	yes

Summary: This package implements in C++/Rcpp various cladogenesis-related calculations that are slow in pure R. These include the calculation of the probability of various scenarios for the inheritance of geographic range at the divergence events on a phylogenetic tree, and other calculations necessary for models which are not continuous-time markov chains (CTMC), but where change instead occurs instantaneously at speciation events. Typically these models must assess the probability of every possible combination of (ancestor state, left descendent state, right descendent state). This means that there are up to (# of states)^3 combinations to investigate, and in biogeographical models, there can easily be hundreds of states, so calculation time becomes an issue. C++

implementation plus clever tricks (many combinations can be eliminated a priori) can greatly speed the computation time over naive R implementations.

CITATION INFO: This package is the result of my Ph.D. research, please cite the package if you use it! Type: `citation(package="cladoRcpp")` to get the citation information.

See also the citation information for the sister packages, `citation(package="rexpokit")` and `citation(package="BioGeoBEARS")`

Further information: In particular, in `cladoRcpp`, functions are implemented to calculate the probability, given a model, of various scenarios for the inheritance of geographic range at speciation events, where the left and right branches may inherit ranges different from each other and different from the ancestor.

The documentation for `rcpp_areas_list_to_states_list`, and `rcpp_calc_anclikes_sp` contain the basic introduction to the logic of ancestral states and cladogenesis probabilities with historical-biogeography models.

The widely-used historical biogeography program `LAGRANGE` (Ree & Smith 2008) has only one cladogenesis model, which is fixed and therefore not subject to inference. LAGRANGE's cladogenesis model gives equal weight/equal probability to all allowed cladogenesis events. LAGRANGE allows:

1. sympatric speciation (copying the ancestral range to descendant ranges), but only for ranges of size=1 area
2. vicariant speciation (the descendant range is divided between the 2 descendant species), but at least one of the descendants must have a ranges of size=1 area
3. sympatric "subset" speciation (one species starts inside the ancestral range, the other inherits the ancestral range); again, one of the descendants must have a ranges of size=1 area

But, another range inheritance scenario is imaginable:

4. founder-event speciation, where one descendant species inherits the ancestral range, and the other species has a range completely outside of the ancestral range

`cladoRcpp` allows specification of these different models, including allowing different weights for the different processes, if users would like to infer the optimal model, rather than simply fixing it ahead of time. The optimization and model choice is done with the help of the sister packages, `rexpokit` and `BioGeoBEARS`.

Note: I began this package with a little bit of code from Rcpp and the various examples that have been written with it, as well as from the following:

1. phyloRcppExamples by Vladimir Minin (https://r-forge.r-project.org/scm/viewvc.php/pkg/phyloRcppExamples/?root=evo and http://markovjumps.blogspot.com/2012/01/packaging-and-exposing-rcpp-functions.html) – which shows how to do phylogenetic operations in C++, accessed with R

2. rcppbugs by Whit Armstrong (https://github.com/armstrtw/rcppbugs and http://cran.r-project.org/web/packages/rcppbugs/i – which does BUGS-style MCMC via C++ functions wrapped in R. It does this much faster than MCMCpack and rjags.

Note

Some starting code borrowed from Rcpp examples, Whit Armstrong's rcppbugs, and Vladimir Minin's phyloRcppExamples.

Author(s)

Nicholas J. Matzke <matzke@berkeley.edu>

References

http://phylo.wikidot.com/biogeobears

Matzke N (2012). "Founder-event speciation in BioGeoBEARS package dramatically improves likelihoods and alters parameter inference in Dispersal-Extinction-Cladogenesis (DEC) analyses." _Frontiers of Biogeography_, *4*(suppl. 1), pp. 210. ISSN 1948-6596, Poster abstract published in the Conference Program and Abstracts of the International Biogeography Society 6th Biannual Meeting, Miami, Florida. Poster Session P10: Historical and Paleo-Biogeography. Poster 129B. January 11, 2013, <URL: http://phylo.wikidot.com/matzke-2013-international-biogeography-society-poster

Ree RH, Moore BR, Webb CO and Donoghue MJ (2005). "A likelihood framework for inferring the evolution of geographic range on phylogenetic trees." _Evolution_, *59*(11), pp. 2299-311. Ree, Richard H Moore, Brian R Webb, Campbell O Donoghue, Michael J Research Support, U.S. Gov't, Non-P.H.S. United States Evolution; international journal of organic evolution Evolution. 2005 Nov;59(11):2299-311., <URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&

Ree RH and Smith SA (2008). "Maximum likelihood inference of geographic range evolution by dispersal, local extinction, and cladogenesis." _Systematic Biology_, *57*(1), pp. 4-14. <URL: http://dx.doi.org/10.1080/10635150701883881>, <URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=

See Also

rcpp_calc_anclikes_sp, rcpp_areas_list_to_states_list, Rcpp, RcppArmadillo

Examples

```
# To get citation information for cladoRcpp, type:
citation(package="cladoRcpp")

# Please also cite the sister packages I created to utilize rexpokit:
# citation(package="rexpokit") # Roger Sidje is a coauthor
                                # of rexpokit and author of
                                # the FORTRAN EXPOKIT
# citation(package="BioGeoBEARS")

library(cladoRcpp)
# Test this first as it causes problems for --gct or --use-valgrind
areas_list = c("A", "B", "C")
areas_list

# Calculate the list of 0-based indices for each possible
# geographic range, i.e. each combination of areas
## Not run:

states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=FALSE)


## End(Not run)
```

```
#############################################################################
# Examples using C++ to speed up the slow step of getting all possible combinations
# of areas (important when when number_of_areas >= 7, as this can mean
# 2^number_of_areas states, and (2^number_of_areas)^2 imaginable descendant pairs
# from each ancestral state.
#############################################################################

#########################################################
# Set up 2 vectors, then convolve them
#########################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
rcpp_convolve(a=ca, b=cb)
#'
# Same as:
convolve(ca, cb, conj=TRUE, type="open")



#########################################################
# Cross-products
#########################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
rcpp_mult2probvect(a=ca, b=cb)

# Same as:
c(ca %o% cb)

# Or:
outer(ca, cb)
c(outer(ca, cb))

# Or:
tcrossprod(ca, cb)
c(tcrossprod(ca, cb))




#############################################################################
# Calculate the number of states (i.e., number of difference geographic ranges,
# i.e. number of different combinations of presence/absence in areas) based on
# the number of areas
#############################################################################
numstates_from_numareas(numareas=3, maxareas=3, include_null_range=FALSE)
numstates_from_numareas(numareas=3, maxareas=3, include_null_range=TRUE)
numstates_from_numareas(numareas=3, maxareas=2, include_null_range=TRUE)
numstates_from_numareas(numareas=3, maxareas=1, include_null_range=TRUE)
numstates_from_numareas(numareas=7, maxareas=7, include_null_range=TRUE)
numstates_from_numareas(numareas=7, maxareas=2, include_null_range=TRUE)
```

```
numstates_from_numareas(numareas=8, maxareas=8, include_null_range=TRUE)
numstates_from_numareas(numareas=8, maxareas=2, include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=20, include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=2, include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=3, include_null_range=TRUE)



#############################################################################
# Generate the list of states based on the list of areas
# And then generate the continuous-time transition matrix (Q matrix)
# for changes that happen along branches
# (the changes that happen at nodes are cladogenesis events)
#############################################################################
# Specify the areas
areas_list = c("A", "B", "C")
areas_list

# Let's try Rcpp_combn_zerostart, in case that is the source of a
# problem found via AddressSanitizer
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=2, maxlim=1e+07)
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=3, maxlim=1e+07)


# Calculate the list of 0-based indices for each possible
# geographic range, i.e. each combination of areas
## Not run:

states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=FALSE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=2,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=1,
include_null_range=TRUE)
states_list


## End(Not run)

# Hard-code the along-branch dispersal and extinction rates
d = 0.2
e = 0.1

# Calculate the dispersal weights matrix and the extinction weights matrix
# Equal dispersal in all directions (unconstrained)
areas = areas_list
distances_mat = matrix(1, nrow=length(areas), ncol=length(areas))
```

```
dmat = matrix(d, nrow=length(areas), ncol=length(areas))
dmat

# Equal extinction probability for all areas
elist = rep(e, length(areas))
elist

# Set up the instantaneous rate matrix (Q matrix, Qmat)
# DON'T force a sparse-style (COO-formatted) matrix here
## Not run:

force_sparse = FALSE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat

# DO force a sparse-style (COO-formatted) matrix here
force_sparse = TRUE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat

## End(Not run)




################################################################################
# Calculate the probability of each (ancestral range) -->
# (Left,Right descendant range pair) directly
################################################################################

######################################################
# Silly example, but which shows the math:
######################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Another silly example, but which shows the normalization effect of specifying
# Rsp_rowsums:
ca_1s = c(1,1,1,1,1)
cb_1s = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))

# Get the Rsp_rowsums (sum across each row; each row=an ancestral state)
```

```
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca_1s, Rcpp_rightprobs=cb_1s,
l=temp_states_indices,  s=0.33, v=0.33, j=0, y=0.33, printmat=TRUE)
Rsp_rowsums

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

######################################################
# Silly example, but which shows the math -- redo with same cladogenesis model,
# specified differently
######################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0.33, v=0.33, j=0, y=0.33, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Another silly example, but which shows the normalization effect of specifying
# Rsp_rowsums:
ca_1s = c(1,1,1,1,1)
cb_1s = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))

# Get the Rsp_rowsums (sum across each row; each row=an ancestral state)
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca_1s, Rcpp_rightprobs=cb_1s,
l=temp_states_indices,  s=0.33, v=0.33, j=0, y=0.33, printmat=TRUE)
Rsp_rowsums

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0.33, v=0.33, j=0, y=0.33,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state



######################################################
# Silly example, but which shows the math -- redo with different cladogenesis model
# (sympatric-copying only, maximum range size of 1)
######################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0, v=0, j=0, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Another silly example, but which shows the normalization effect of specifying
# Rsp_rowsums:
ca_1s = c(1,1,1,1,1)
cb_1s = c(1,1,1,1,1)
```

```
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))

# Get the Rsp_rowsums (sum across each row; each row=an ancestral state)
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca_1s, Rcpp_rightprobs=cb_1s,
l=temp_states_indices,  s=0, v=0, j=0, y=1, printmat=TRUE)
Rsp_rowsums

# Note that you get NaNs because some of your states (2 areas) are impossible on
# this model
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0, v=0, j=0, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state



######################################################
# Silly example, but which shows the math -- redo with different
# cladogenesis model (BayArea, sympatric-copying only)
######################################################
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
numareas = 3
maxent01y = matrix(0, nrow = numareas, ncol = numareas)
maxent01y[, 1] = seq(1, numareas)
maxent01y[2:3, 2] = seq(2, numareas)
maxent01y[3, 3] = seq(3, numareas)

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0, v=0, j=0, y=1, maxent01y=maxent01y,
max_minsize_as_function_of_ancsize=rep(3,numareas), printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Another silly example, but which shows the normalization effect of specifying
# Rsp_rowsums:
ca_1s = c(1,1,1,1,1)
cb_1s = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))

# Get the Rsp_rowsums (sum across each row; each row=an ancestral state)
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca_1s, Rcpp_rightprobs=cb_1s,
l=temp_states_indices, s=0, v=0, j=0, y=1, maxent01y=maxent01y,
max_minsize_as_function_of_ancsize=rep(3,numareas), printmat=TRUE)
Rsp_rowsums

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0, v=0, j=0, y=1, maxent01y=maxent01y,
max_minsize_as_function_of_ancsize=rep(3,numareas), printmat=TRUE)
condlike_of_data_for_each_ancestral_state
```

```
#######################################################
# Actual example
#######################################################
# When ca & cb are 1s, and s, v, j, y are 1s or 0s:
# ...this shows how many possible descendant pairs are possible from each
# possible ancestor, under the model
# i.e., how many specific cladogenesis scenarios are possible from each
# possible ancestor
# This is the LAGRANGE model
ca = c(1,1,1,1,1)
cb = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
Rsp_rowsums

# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
# WITHOUT the weights correction
ca = c(0.2,0.2,0.2,0.2,0.2)
cb = c(0.2,0.2,0.2,0.2,0.2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# WITH the weights correction
ca = c(0.2,0.2,0.2,0.2,0.2)
cb = c(0.2,0.2,0.2,0.2,0.2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state



# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches are NOT equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state
```

```
# WITH the weights correction
# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches are NOT equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state




#########################################################
# Actual example -- for another model (allowing jump dispersal equal probability
# with the rest)
#########################################################
# When ca & cb are 1s, and s, v, j, y are 1s or 0s:
# ...this shows how many possible descendant pairs are possible from each possible
# ancestor, under the model
# i.e., how many specific cladogenesis scenarios are possible from each possible
# ancestor
# This is the LAGRANGE model
ca = c(1,1,1,1,1)
cb = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
Rsp_rowsums = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=1, v=1, j=1, y=1, printmat=TRUE)
Rsp_rowsums

# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
# WITHOUT the weights correction
ca = c(0.2,0.2,0.2,0.2,0.2)
cb = c(0.2,0.2,0.2,0.2,0.2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
```

```
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# WITH the weights correction
ca = c(0.2,0.2,0.2,0.2,0.2)
cb = c(0.2,0.2,0.2,0.2,0.2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state



# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches are NOT equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# WITH the weights correction
# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state

# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches are NOT equal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1, Rsp_rowsums=Rsp_rowsums,
printmat=TRUE)
condlike_of_data_for_each_ancestral_state
```

```
#######################################################
# Calculate the sums of each row (i.e. for each ancestral state) --
# changes only based on the model
#######################################################
# Standard LAGRANGE model
# Rcpp_leftprobs=ca, Rcpp_rightprobs=cb are irrelevant except for length,
# rcpp_calc_anclikes_sp_rowsums() actually treats them as arrays of 1s
# if s, v, j, y are 1s or 0s, then Rsp_rowsums = counts of the events
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
Rsp_rowsums

# Standard LAGRANGE model, adding jump dispersal
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=1, v=1, j=1, y=1, printmat=TRUE)
Rsp_rowsums

# Same models, parameterized differently
# Allowing jump dispersal to areas outside of the ancestral range
Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=0.5, v=0.5, j=0, y=0.5, printmat=TRUE)
Rsp_rowsums

# Allowing jump dispersal to areas outside of the ancestral range
Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=0.5, v=0.5, j=0.5, y=0.5, printmat=TRUE)
Rsp_rowsums




#######################################################
# The relative weights of the different types of cladogenesis events doesn't matter,
# if the correction factor is included
#######################################################

# LAGRANGE+founder-event speciation
# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches are NOT equal
```

```
# s,v,j,y weights set to 1
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
uncorrected_condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1,
printmat=TRUE)
uncorrected_condlike_of_data_for_each_ancestral_state

Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=1, v=1, j=1, y=1, printmat=TRUE)
Rsp_rowsums

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=1, y=1,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state


# s,v,j,y weights set to 0.5
ca = c(0.05,0.1,0.15,0.2,0.5)
cb = c(0.05,0.1,0.15,0.2,0.5)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
uncorrected_condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, l=temp_states_indices, s=0.5, v=0.5, j=0.5,
y=0.5, printmat=TRUE)
uncorrected_condlike_of_data_for_each_ancestral_state

Rsp_rowsums = rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs=ca, Rcpp_rightprobs=cb,
l=temp_states_indices, s=0.5, v=0.5, j=0.5, y=0.5, printmat=TRUE)
Rsp_rowsums

condlike_of_data_for_each_ancestral_state = rcpp_calc_anclikes_sp(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=0.5, v=0.5, j=0.5, y=0.5,
Rsp_rowsums=Rsp_rowsums, printmat=TRUE)
condlike_of_data_for_each_ancestral_state







######################################################
# For large state spaces (many areas, a great many possible geographic ranges i.e.
# combinations of areas),
# rcpp_calc_anclikes_sp() gets slow, even with C++ implementation, as it loops
# through every possible combination
# of ancestral and descendant states.  rcpp_calc_anclikes_sp_COOprobs() is a partial
# speedup which takes various shortcuts.
#
# Instead of having the weights/probabilites represented internally, and producing
# the conditional likelihoods as output,
```

```
# rcpp_calc_anclikes_sp_COOprobs() produces 3 lists, giving the coordinates of
# nonzero cells in the transition matrix.
#
# List #1: 0-based index of states on the Left branch, for each of the ancestral
# states
# List #2: 0-based index of states on the Right branch, for each of the ancestral
# states
# List #3: Weight of each transition, for each of the ancestral states
########################################################


# Silly example, but which shows the math:
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
list_weights_of_transitions = rcpp_calc_anclikes_sp_COOprobs(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
list_weights_of_transitions

# List #1: 0-based index of states on the Left branch, for each of the ancestral states
# List #2: 0-based index of states on the Right branch, for each of the ancestral
# states
# List #3: Weight of each transition, for each of the ancestral states



# Get the Rsp_rowsums (sums of the rows of the cladogenesis P matrix)
# Set the weights to 1
ca = c(1,1,1,1,1)
cb = c(1,1,1,1,1)
COO_probs_list_for_rowsums = rcpp_calc_anclikes_sp_COOprobs(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
COO_probs_list_for_rowsums
Rsp_rowsums = sapply(X=COO_probs_list_for_rowsums[[3]], FUN=sum)
Rsp_rowsums


# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
ca = c(0.2,0.2,0.2,0.2,0.2)
cb = c(0.2,0.2,0.2,0.2,0.2)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
COO_weights_list = rcpp_calc_anclikes_sp_COOprobs(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
COO_weights_list

COO_weights_list_rowsums = sapply(X=COO_probs_list_for_rowsums[[3]], FUN=sum)
COO_weights_list_rowsums

# To see the transitional probabilities for each ancestral state, under the model:
COO_format_transition_probability_matrix = COO_probs_list_for_rowsums

for (i in 1:length(COO_format_transition_probability_matrix[[3]]))
```

```
{
COO_format_transition_probability_matrix[[3]][[i]] =
 COO_format_transition_probability_matrix[[3]][[i]] / COO_weights_list_rowsums[[i]]
}
COO_format_transition_probability_matrix

# And you can see that the probabilities now sum to 1 for each row
sapply(X=COO_format_transition_probability_matrix[[3]], FUN=sum)



# The following is what you would get for the conditional likelihoods of the
# data given each ancestral state, WITHOUT making each row
# of the transition matrix sum to 1
uncorrected_COO_condlikes_list = rcpp_calc_anclikes_sp_using_COOprobs(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, RCOO_left_i_list=COO_weights_list[[1]],
RCOO_right_j_list=COO_weights_list[[2]], RCOO_probs_list=COO_weights_list[[3]],
Rsp_rowsums=rep(1,length(ca)), printmat=TRUE)
uncorrected_COO_condlikes_list

# This is what you get if you correct, so that each row sums to 1,
# using the sums of the rows to normalize
corrected_COO_condlikes_list = rcpp_calc_anclikes_sp_using_COOprobs(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, RCOO_left_i_list=COO_weights_list[[1]],
RCOO_right_j_list=COO_weights_list[[2]], RCOO_probs_list=COO_weights_list[[3]],
Rsp_rowsums=COO_weights_list_rowsums, printmat=TRUE)
corrected_COO_condlikes_list



################################################################################
# rcpp_calc_anclikes_sp_COOweights_faster():
# An even faster method "intelligently" looks for allowed transitions with
# nonzero weights
# The output is stored in 4 lists / columns in COO_weights_columnar:
# List #1. 0-based index of ancestral states / geographic ranges
# List #2. 0-based index of Left descendant states / geographic ranges
# List #3. 0-based index of Right descendant states / geographic ranges
# List #4. Weight (or probability, if each weight has been divided by the sum
# of the weights for the row) of the
#    transition specified by that cell.
################################################################################



# Get the Rsp_rowsums (sums of the rows of the cladogenesis P matrix)
# Set the weights to 1
ca = c(1,1,1,1,1) # ca and cb don't matter here, since we are just calculating
# the weights
```

```
cb = c(1,1,1,1,1)
temp_states_indices = list(c(0), c(1), c(2), c(0,1), c(1,2))
COO_weights_columnar = rcpp_calc_anclikes_sp_COOweights_faster(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1,
j=0, y=1, printmat=TRUE)
COO_weights_columnar

# List #1. 0-based index of ancestral states / geographic ranges
# List #2. 0-based index of Left descendant states / geographic ranges
# List #3. 0-based index of Right descendant states / geographic ranges
# List #4. Weight (or probability, if each weight has been divided by the
# sum of the weights for the row) of the
#    transition specified by that cell.

# Calculate the sums of the weights for each row/ancestral state
numstates = 1+max(sapply(X=COO_weights_columnar, FUN=max)[1:3])
Rsp_rowsums = rcpp_calc_rowsums_for_COOweights_columnar(
COO_weights_columnar=COO_weights_columnar, numstates=numstates)
Rsp_rowsums



# Silly example, but which shows the math:
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)

# WITHOUT using appropriate correction (correction = dividing by the
# sum of the weights for each row)
uncorrected_condlikes_list = rcpp_calc_splitlikes_using_COOweights_columnar(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, COO_weights_columnar=COO_weights_columnar,
Rsp_rowsums=rep(1,numstates), printmat=TRUE)
uncorrected_condlikes_list

# WITH using appropriate correction (correction = dividing by the sum
# of the weights for each row)
corrected_condlikes_list = rcpp_calc_splitlikes_using_COOweights_columnar(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, COO_weights_columnar=COO_weights_columnar,
Rsp_rowsums, printmat=TRUE)
corrected_condlikes_list




# Calculate likelihoods of ancestral states if probabilities of each state
# at the base of the left and right branches ARE equal
# Get the Rsp_rowsums (sums of the rows of the cladogenesis P matrix)
# Set the weights to 1
ca = c(0.2,0.2,0.2,0.2,0.2) # ca and cb don't matter here, since we are just
# calculating the weights
cb = c(0.2,0.2,0.2,0.2,0.2)
COO_weights_columnar = rcpp_calc_anclikes_sp_COOweights_faster(Rcpp_leftprobs=ca,
Rcpp_rightprobs=cb, l=temp_states_indices, s=1, v=1, j=0, y=1, printmat=TRUE)
COO_weights_columnar
```

```
# List #1. 0-based index of ancestral states / geographic ranges
# List #2. 0-based index of Left descendant states / geographic ranges
# List #3. 0-based index of Right descendant states / geographic ranges
# List #4. Weight (or probability, if each weight has been divided by the
# sum of the weights for the row) of the
#    transition specified by that cell.

# Calculate the sums of the weights for each row/ancestral state
numstates = 1+max(sapply(X=COO_weights_columnar, FUN=max)[1:3])
Rsp_rowsums = rcpp_calc_rowsums_for_COOweights_columnar(
COO_weights_columnar=COO_weights_columnar, numstates=numstates)
Rsp_rowsums


# WITHOUT using appropriate correction (correction = dividing by
# the sum of the weights for each row)
uncorrected_condlikes_list = rcpp_calc_splitlikes_using_COOweights_columnar(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, COO_weights_columnar=COO_weights_columnar,
Rsp_rowsums=rep(1,numstates), printmat=TRUE)
uncorrected_condlikes_list

# WITH using appropriate correction (correction = dividing by the sum of the
# weights for each row)
corrected_condlikes_list = rcpp_calc_splitlikes_using_COOweights_columnar(
Rcpp_leftprobs=ca, Rcpp_rightprobs=cb, COO_weights_columnar=COO_weights_columnar,
Rsp_rowsums, printmat=TRUE)
corrected_condlikes_list
```

`areas_list_to_states_list_old` *Convert a list of areas to a list of geographic ranges (states); original R version*

Description

This is the original R version of the function which converts a list of possible areas to a list of all possible states (geographic ranges). This gets slow for large numbers of areas.

Usage

```
areas_list_to_states_list_old(areas = c("A", "B", "C"),
  maxareas = length(areas), include_null_range = TRUE,
  split_ABC = TRUE)
```

Arguments

`areas`	a list of areas (character or number; the function converts these to numbers, starting with 0)

maxareas	maximum number of areas in this analyses
include_null_range	TRUE or FALSE, should the NULL range be included in the possible states? (e.g., LAGRANGE default is yes)
split_ABC	TRUE or FALSE If TRUE the output will consist of a list of lists (c("A","B","C"), c("A","B"), c("A","D"), etc.); if FALSE, the list of areas will be collapsed ("ABC", "AB", "AD", etc.).

Details

The function is mostly replaced by rcpp_areas_list_to_states_list in optimized code, but is still used in some places for display purposes.

Value

states_list A list of the states.

Note

No notes.

Author(s)

Nicholas J. Matzke <matzke@berkeley.edu>

References

http://phylo.wikidot.com/matzke-2013-international-biogeography-society-poster https://code.google.com/p/lagrange/ #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

See Also

numstates_from_numareas, rcpp_areas_list_to_states_list

Examples

```
areas = c("A","B","C")
areas_list_to_states_list_old(areas=areas, maxareas=length(areas),
include_null_range=TRUE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=length(areas),
include_null_range=TRUE, split_ABC=FALSE)
areas_list_to_states_list_old(areas=areas, maxareas=length(areas),
include_null_range=FALSE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=length(areas),
include_null_range=FALSE, split_ABC=FALSE)
areas_list_to_states_list_old(areas=areas, maxareas=2,
include_null_range=TRUE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=2,
include_null_range=TRUE, split_ABC=FALSE)
areas_list_to_states_list_old(areas=areas, maxareas=2,
```

```
include_null_range=FALSE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=2,
include_null_range=FALSE, split_ABC=FALSE)
areas_list_to_states_list_old(areas=areas, maxareas=1,
include_null_range=TRUE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=1,
include_null_range=TRUE, split_ABC=FALSE)
areas_list_to_states_list_old(areas=areas, maxareas=1,
include_null_range=FALSE, split_ABC=TRUE)
areas_list_to_states_list_old(areas=areas, maxareas=1,
include_null_range=FALSE, split_ABC=FALSE)
```

`numstates_from_numareas` *Calculate the number of states, given a certain number of areas*

Description

This function calculates the number of discrete states that are needed to represent the possible combinations of presence and absence in a set of discrete areas. The number of states is a function of the number of areas, and the maximum allowed range size (in number of areas) of a species.

Usage

```
numstates_from_numareas(numareas = 3, maxareas = numareas,
  include_null_range = FALSE)
```

Arguments

`numareas`	The number of areas in the analysis.
`maxareas`	The maximum number of areas that any single species/lineage can occupy.
`include_null_range`	If FALSE (default), the null range is not included in the count. If TRUE, the null range is included, adding +1 to the count of the states.

Details

For example, with 3 areas (A, B, C), there are 8 possible states, if a null range is allowed (null, A, B, C, AB, BC, AC, ABC). If the maximum range size is 2 areas, then there are only 7 possible states.

The formula for the number of geographic states, based on the number of areas (N), is the sum of N choose k, from k=1 to m (maximum range size)

$$s = \sum_{k=1}^{m} \binom{N}{k}$$

This equation assumes that the null range (a species lives in no areas, i.e. is extinct) is not allowed. In the LAGRANGE program of *ReeSmith2008*), the null range is included in the transition matrix,

and thus this is one more state. This situation is represented in `numstates_from_numareas` by setting `include_null_range=TRUE`.

Users might manually remove states from the states list, if prior information indicates that some configurations of presence/absence in areas are impossible as geographic ranges for species. If so, they should manually subtract from the number of states.

Value

`nstates` Number of states

Author(s)

Nicholas Matzke `<matzke@berkeley.edu>`

See Also

`convolve` #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

Examples

```
numstates_from_numareas(numareas=3, maxareas=3,
include_null_range=FALSE)
numstates_from_numareas(numareas=3, maxareas=3,
include_null_range=TRUE)
numstates_from_numareas(numareas=3, maxareas=2,
include_null_range=TRUE)
numstates_from_numareas(numareas=3, maxareas=1,
include_null_range=TRUE)
numstates_from_numareas(numareas=7, maxareas=7,
include_null_range=TRUE)
numstates_from_numareas(numareas=7, maxareas=2,
include_null_range=TRUE)
numstates_from_numareas(numareas=8, maxareas=8,
include_null_range=TRUE)
numstates_from_numareas(numareas=8, maxareas=2,
include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=20,
include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=2,
include_null_range=TRUE)
numstates_from_numareas(numareas=20, maxareas=3,
include_null_range=TRUE)
```

`rcpp_areas_list_to_states_list` *Make a list of 0-based indices of possible combinations of input areas*

Description

Given a list of areas (actually a list of anything; all that is important is the length of the list) `rcpp_areas_list_to_states_list` calculates all possible combinations of these areas, listing them by the 0-based indices that specify the position of each area in the list.

Usage

```
rcpp_areas_list_to_states_list(areas = c("A", "B", "C"),
  maxareas = length(areas), include_null_range = TRUE)
```

Arguments

`areas`	a list of areas (character or number; the function converts these to numbers, starting with 0)
`maxareas`	maximum number of areas in this analyses
`include_null_range`	`TRUE` or `FALSE`, should the `NULL` range be included in the possible states? (e.g., `LAGRANGE` default is yes)

Details

Using 0-based indexing is convenient in the C++ code called by the other functions, rather than having to keep track of the various people might label their areas (names, abbreviations, letters, numbers).

As in LAGRANGE (Ree & Smith 2008), the maximum range size (i.e. the maximum number of areas in a range) can be specified by the user. Having a smaller maximum range size drastically reduces the number of states, and thus the size of the transition matrix and the cladogenesis matrix.

Value

`R_states_list` A list of the states, where each state is a list of areas in the form of 0-based indices

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

`numstates_from_numareas`, `areas_list_to_states_list_old` #bibliography /Dropbox/_njm/__packages/cladoRcpp_set
@cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

Examples

```
# Specify the areas
areas_list = c("A", "B", "C")
areas_list

# Let's try Rcpp_combn_zerostart, in case that is the source of a
# problem found via AddressSanitizer
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=2, maxlim=1e+07)
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=3, maxlim=1e+07)


## Not run:

# Calculate the list of 0-based indices for each possible geographic range, i.e.
# each combination of areas
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=FALSE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=2,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=1,
include_null_range=TRUE)
states_list


## End(Not run)
```

---

rcpp_calc_anclikes_sp *Calculate probability of ancestral states below a speciation event, given probabilities of the states on each descendant branch*

Description

This function, given parameters on the Relative weight of different geographic range inheritance scenarios at cladogenesis (speciation) events, calculates the probability of each possible ancestral state given the probabilities of each possible combination of tip states.

Usage

```
rcpp_calc_anclikes_sp(Rcpp_leftprobs, Rcpp_rightprobs, l, s = 1, v = 1,
  j = 0, y = 1, dmat = NULL, maxent01s = NULL, maxent01v = NULL,
  maxent01j = NULL, maxent01y = NULL,
  max_minsize_as_function_of_ancsize = NULL, Rsp_rowsums = rep(1,
  length(Rcpp_leftprobs)), printmat = FALSE)
```

Arguments

Rcpp_leftprobs	Probabilities of the states at the base of the left descendant branch
Rcpp_rightprobs	Probabilities of the states at the base of the right descendant branch
l	List of state indices (0-based)
s	Relative weight of sympatric "subset" speciation. Default s=1 mimics LAGRANGE model.
v	Relative weight of vicariant speciation. Default v=1 mimics LAGRANGE model.
j	Relative weight of "founder event speciation"/jump speciation. Default j=0 mimics LAGRANGE model.
y	Relative weight of fully sympatric speciation (range-copying). Default y=1 mimics LAGRANGE model.
dmat	If given, a matrix of rank numareas giving multipliers for the probability of each dispersal event between areas. Default NULL, which sets every cell of the dmat matrix to value 1. Users may construct their own parameterized dmat (for example, making dmat a function of distance) for inclusion in ML or Bayesian analyses.
maxent01s	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a subset-sympatric speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01v	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a vicariance speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01j	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a founder-event speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01y	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a full-sympatric (range-copying) speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
max_minsize_as_function_of_ancsize	If given, any state with a range larger that this value will be given a probability of zero (for the branch with the smaller rangesize). This means that not every possible combination of ranges has to be checked, which can get very slow for large state spaces.
Rsp_rowsums	A vector of size (numstates) giving the sum of the relative probabilites of each combination of descendant states, assuming the probabilities of the left- and right-states are all equal (set to 1). This is thus the sum of the weights, and dividing by this normalization vector means that the each row of the speciation probability matrix will sum to 1. Default assumes the weights sum to 1 but this is not usually the case. Rsp_rowsums need only be calculated once

per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.

`printmat` Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Details

The Python/C++ program LAGRANGE (Ree & Smith 2008) gives a fixed equal probability to each range-inheritance scenario it allows:

(1) sympatric speciation with 1 area (e.g. A –> A,A);
(2) sympatric speciation where one species inherits the ancestral range, and the other inherits a 1-area subset of the ancestral range (e.g. ABC –> ABC,B);
(3) vicariant speciation with one daughter occupying an area of size 1 (e.g. ABCD –> ACD,B)

For example, if the ancestral range is ABC, the possible daughters are:

(Left, Right)

Vicariance: A,BC AB,C AC,B BC,A C,AB B,AC

Sympatric subset: A,ABC B,ABC C,ABC ABC,A ABC,B ABC,C

There are 12 possibilities, so LAGRANGE would give each a probability of 1/12, conditional on the ancestor having range ABC. All other imaginable scenarios are given probability 0 – e.g., sympatric speciation of a widespread range (ABC –> ABC,ABC), or jump dispersal leading to founder-event speciation (ABC –> ABC,D).

In `BioGeoBEARS`, the relative probability (or weight) of these categories is set by the `s` (sympatric-subset), `v` (vicariance), `j` (jump/founder-event), and `y` (sympatric-range-copying) parameters. These parameters do not have to sum to 1, they just give the *relative* weight of an event of each type. E.g., if `s=1`, `v=1`, `j=0`, `y=1`, then each allowed sympatric-range-copying, sympatric-subset, and vicariance event is given equal probability (this is the LAGRANGE cladogenesis model) .

The `rcpp_calc_anclikes_sp` function gets slow for large state spaces, as every possible combination of states at Left and Right branches is checked. Even in C++ this will get slow, as the `(number of states) = 2^(number of areas)`, and as the number of possible combinations of (ancestor, left,right) states is `(number of states)*(number of states)*(number of states)`.

Note: the `maxent` parameters allow the user to specify the probability distribution for different range sizes of the smaller-ranged descendant lineage. The defaults set these parameters so that the LAGRANGE model is implemented (the smaller descendant always has range size 1).

See `rcpp_calc_anclikes_sp_COOprobs` and `rcpp_calc_anclikes_sp_COOweights_faster` for successively faster solutions to this problem.

This is the byte-compiled version of `rcpp_calc_anclikes_sp_prebyte`. `rcpp_calc_anclikes_sp` is byte-compiled, which (might) make it faster.

For information on byte-compiling, see http://www.r-statistics.com/2012/04/speed-up-your-r-code-using-a-ju and `cmpfun` in the `compiler` package.

Value

`prob_ancestral_states` The probabilities of the ancestral states.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp, rcpp_calc_anclikes_sp_COOprobs, rcpp_calc_anclikes_sp_COOweights_faster
#bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013
@cite Matzke_2014 @cite ReeSmith2008

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

---

rcpp_calc_anclikes_sp_COOprobs

Faster version of rcpp_calc_anclikes_sp

Description

This function is a faster version of rcpp_calc_anclikes_sp. Like rcpp_calc_anclikes_sp, this function calculates the conditional probability of every allowed combination of ancestral range, left descendent range, and right descendent range.

Usage

```
rcpp_calc_anclikes_sp_COOprobs(Rcpp_leftprobs, Rcpp_rightprobs, l, s = 1,
  v = 1, j = 0, y = 1, dmat = NULL, maxent01s = NULL,
  maxent01v = NULL, maxent01j = NULL, maxent01y = NULL,
  max_minsize_as_function_of_ancsize = NULL, printmat = TRUE)
```

Arguments

Rcpp_leftprobs	Probabilities of the states at the base of the left descendant branch
Rcpp_rightprobs	Probabilities of the states at the base of the right descendant branch
l	List of state indices (0-based)
s	Relative weight of sympatric "subset" speciation. Default s=1 mimics LAGRANGE model.
v	Relative weight of vicariant speciation. Default v=1 mimics LAGRANGE model.
j	Relative weight of "founder event speciation"/jump speciation. Default j=0 mimics LAGRANGE model.

`y` Relative weight of fully sympatric speciation (range-copying). Default `y=1` mimics LAGRANGE model.

`dmat` If given, a matrix of rank numareas giving multipliers for the probability of each dispersal event between areas. Default NULL, which sets every cell of the `dmat` matrix to value 1. Users may construct their own parameterized `dmat` (for example, making `dmat` a function of distance) for inclusion in ML or Bayesian analyses.

`maxent01s` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a subset-sympatric speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01v` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a vicariance speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01j` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a founder-event speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01y` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a full-sympatric (range-copying) speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`max_minsize_as_function_of_ancsize`
If given, any state with a range larger that this value will be given a probability of zero (for the branch with the smaller rangesize). This means that not every possible combination of ranges has to be checked, which can get very slow for large state spaces.

`printmat` Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Details

This function improves upon `rcpp_calc_anclikes_sp` by returning a COO-like list of the nonzero cells in the transition matrix for the speciation event.

(COO = Coordinate list format for a matrix, see http://en.wikipedia.org/wiki/Sparse_matrix#Coordinate_list_.28COO.29

Whereas a COO-formatted square matrix stores, for each nonzero cell, the row #, column #, and cell value, `rcpp_calc_anclikes_sp` returns lists containing, for each nonzero cell:

1. 0-based index of the ancestral state
2. 0-based index of the left state
3. 0-based index of the right state
4. Value of the specified nonzero cell

Time savings over rcpp_calc_anclikes_sp are realized by skipping many ancestor/descendent combinations which are impossible transitions on the model, and neither recording, nor storing, nor passing them. This becomes important with large state spaces.

Value

list_weights_of_transitions A list of 3 lists. Each list has (numstates) items, representing the ancestral states. List #1 gives the 0-based state index for the nonzero left descendents of each ancestral state. List #2 gives the 0-based state index for the nonzero right descendents of each ancestral state. List #3 gives the weight of each nonzero transition from each ancestral state. Summing these weights within each ancestral state for list #3 gives the total of the weights for each ancestral state. Dividing the weights by the sum of weights gives the conditional probability of each descendent state, conditional on the ancestral state. These conditional probabilities need only be calculated once per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp, rcpp_calc_anclikes_sp_COOprobs, rcpp_calc_anclikes_sp_COOweights_faster
#bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013
@cite Matzke_2014 @cite ReeSmith2008

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

---

rcpp_calc_anclikes_sp_COOweights_faster

Even faster version of rcpp_calc_anclikes_sp

Description

This function improves on rcpp_calc_anclikes_sp and rcpp_calc_anclikes_sp_COOprobs. In addition to the compressed COO-like storage format, the internal C++ code here explicitly enumerates the allowed transitions, rather than searching through every possibility and testing whether or not it is allowed. This appears to scale well to very large state spaces.

Usage

```
rcpp_calc_anclikes_sp_COOweights_faster(Rcpp_leftprobs, Rcpp_rightprobs, l,
  s = 1, v = 1, j = 0, y = 1, dmat = NULL, maxent01s = NULL,
  maxent01v = NULL, maxent01j = NULL, maxent01y = NULL,
  max_minsize_as_function_of_ancsize = NULL, printmat = TRUE,
  m = NULL, m_null_range = TRUE, jts_matrix = NULL)
```

Arguments

Rcpp_leftprobs	Probabilities of the states at the base of the left descendant branch
Rcpp_rightprobs	Probabilities of the states at the base of the right descendant branch
l	List of state indices (0-based)
s	Relative weight of sympatric "subset" speciation. Default s=1 mimics LAGRANGE model.
v	Relative weight of vicariant speciation. Default v=1 mimics LAGRANGE model.
j	Relative weight of "founder event speciation"/jump speciation. Default j=0 mimics LAGRANGE model.
y	Relative weight of fully sympatric speciation (range-copying). Default y=1 mimics LAGRANGE model.
dmat	If given, a matrix of rank numareas giving multipliers for the probability of each dispersal event between areas. Default NULL, which sets every cell of the dmat matrix to value 1. Users may construct their own parameterized dmat (for example, making dmat a function of distance) for inclusion in ML or Bayesian analyses.
maxent01s	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a subset-sympatric speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01v	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a vicariance speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01j	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a founder-event speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
maxent01y	Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a full-sympatric (range-copying) speciation event. Default is NULL, which means the script will set up the LAGRANGE model (one descendent always has range size 1).
max_minsize_as_function_of_ancsize	If given, any state with a range larger that this value will be given a probability of zero (for the branch with the smaller rangesize). This means that not every possible combination of ranges has to be checked, which can get very slow for large state spaces.

printmat	Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)
m	This is a vector of rate/weight multipliers for dispersal, conditional on the values of some (non-biogeographical) trait. For example, one might hypothesize that flight/flightlessness effects dispersal probability, and manually put a multiplier of 0.001 on the flightlessness state. Or, one might attempt to estimate this. The strategy used in cladoRcpp is to expand the default cladogenetic rate matrix by length(m) times. I.e., if *m* is not NULL, then loop through the values of *m* and apply the multipliers to *d* (and *j*, and *a*) events. Default is NULL.
m_null_range	Is the null range included in the state space in the general analysis? (The function needs to know this, when there are traits, to index the state space correctly.)
jts_matrix	A numtraits x numtraits matrix containing the proportions for trait transitions during j events. E.g., for a sudden switch from trait 1 (flight) to trait 2 (flightlessness) during a jump event.

Details

This should be faster, i.e. by look for each type of event individually.

Returns results as 4 columns: ancestral index, left index, right index, conditional probability given ancestral states (assuming likelihood of descendants is 1). Indexes are 0-based.

Keep in mind that cladogenesis matrices exclude the null state (a range of 0 areas), so if your states list starts with the null range (as is typical/default in DEC-style models) then to get the R 1-based state indices requires e.g. COO_weights_columnar[[1]] + 2.

When the calculation is run at each node in the tree, all that is required is one pass through the COO-like array, with the downpassed probabilities of the states on the left and right branches multiplied by the probability column.

Value

COO_weights_columnar Transition weights matrix in COO-like format as 4 columns: ancestral index, left index, right index, and weight of the specified transition. Indexes are 0-based. Keep in mind that cladogenesis matrices exclude the null state (a range of 0 areas), so if your states list starts with the null range (as is typical/default in DEC-style models) then to get the R 1-based state indices requires e.g. COO_weights_columnar[[1]] + 2.

Dividing the weights by the sum of the weights for a particular ancestral state yields the conditional probabilities of each transition at the speciation event. (assuming likelihood of descendants is 1).

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp, rcpp_calc_anclikes_sp_COOprobs, rcpp_calc_anclikes_sp_COOweights_faster

rcpp_calc_anclikes_sp #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

`rcpp_calc_anclikes_sp_prebyte` *Calculate probability of ancestral states below a speciation event, given probabilities of the states on each descendant branch*

Description

This is the pre-byte compiled version of `rcpp_calc_anclikes_sp`. `rcpp_calc_anclikes_sp` is byte-compiled, which (might) make it faster. See `rcpp_calc_anclikes_sp` for full description and help.

Usage

```
rcpp_calc_anclikes_sp_prebyte(Rcpp_leftprobs, Rcpp_rightprobs, l, s = 1,
  v = 1, j = 0, y = 1, dmat = NULL, maxent01s = NULL,
  maxent01v = NULL, maxent01j = NULL, maxent01y = NULL,
  max_minsize_as_function_of_ancsize = NULL, Rsp_rowsums = rep(1,
  length(Rcpp_leftprobs)), printmat = FALSE)
```

Arguments

`Rcpp_leftprobs`	Probabilities of the states at the base of the left descendant branch
`Rcpp_rightprobs`	Probabilities of the states at the base of the right descendant branch
`l`	List of state indices (0-based)
`s`	Relative weight of sympatric "subset" speciation. Default `s=1` mimics LAGRANGE model.
`v`	Relative weight of vicariant speciation. Default `v=1` mimics LAGRANGE model.
`j`	Relative weight of "founder event speciation"/jump speciation. Default `j=0` mimics LAGRANGE model.
`y`	Relative weight of fully sympatric speciation (range-copying). Default `y=1` mimics LAGRANGE model.
`dmat`	If given, a matrix of rank numareas giving multipliers for the probability of each dispersal event between areas. Default NULL, which sets every cell of the `dmat` matrix to value 1. Users may construct their own parameterized `dmat` (for example, making `dmat` a function of distance) for inclusion in ML or Bayesian analyses.

`maxent01s` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a subset-sympatric speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01v` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a vicariance speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01j` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a founder-event speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01y` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a full-sympatric (range-copying) speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`max_minsize_as_function_of_ancsize`
If given, any state with a range larger that this value will be given a probability of zero (for the branch with the smaller rangesize). This means that not every possible combination of ranges has to be checked, which can get very slow for large state spaces.

`Rsp_rowsums` A vector of size (numstates) giving the sum of the relative probabilites of each combination of descendant states, assuming the probabilities of the left- and right-states are all equal (set to 1). This is thus the sum of the weights, and dividing by this normalization vector means that the each row of the speciation probability matrix will sum to 1. Default assumes the weights sum to 1 but this is not usually the case. Rsp_rowsums need only be calculated once per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.

`printmat` Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Details

This function gets slow for large state spaces.

For information on byte-compiling, see http://www.r-statistics.com/2012/04/speed-up-your-r-code-using-a-ju and `cmpfun` in the `compiler` package.

Value

`prob_ancestral_states` The probabilities of the ancestral states.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

---

rcpp_calc_anclikes_sp_rowsums

Calculate the number of cladogenesis events of nonzero probability for each ancestral state

---

Description

This function takes the list of possible states (*l*), and the parameters of a cladogenesis model (*s*, *v*, *j*, *y*) (which are the relative weights of each of type of cladogenic range inheritance event) and, for each ancestral state, sums the weights of allowed descendant events. Dividing the weights in each row, by the sum of the weights for that row, provides the absolute probabilities of each transition, conditional on the ancestral state for that row.

Usage

```
rcpp_calc_anclikes_sp_rowsums(Rcpp_leftprobs, Rcpp_rightprobs, l, s = 1,
  v = 1, j = 0, y = 1, dmat = NULL, maxent01s = NULL,
  maxent01v = NULL, maxent01j = NULL, maxent01y = NULL,
  max_minsize_as_function_of_ancsize = NULL, printmat = TRUE)
```

Arguments

Rcpp_leftprobs	Probabilities of the states at the base of the left descendant branch
Rcpp_rightprobs	Probabilities of the states at the base of the right descendant branch
l	List of state indices (0-based)
s	Relative weight of sympatric "subset" speciation. Default s=1 mimics LAGRANGE model.
v	Relative weight of vicariant speciation. Default v=1 mimics LAGRANGE model.
j	Relative weight of "founder event speciation"/jump speciation. Default j=0 mimics LAGRANGE model.
y	Relative weight of fully sympatric speciation (range-copying). Default y=1 mimics LAGRANGE model.

`dmat` If given, a matrix of rank numareas giving multipliers for the probability of each dispersal event between areas. Default NULL, which sets every cell of the `dmat` matrix to value 1. Users may construct their own parameterized `dmat` (for example, making `dmat` a function of distance) for inclusion in ML or Bayesian analyses.

`maxent01s` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a subset-sympatric speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01v` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a vicariance speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01j` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a founder-event speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`maxent01y` Matrix giving the relative weight of each possible descendant rangesize for the smaller range, for a given ancestral rangesize, for a full-sympatric (range-copying) speciation event. Default is `NULL`, which means the script will set up the LAGRANGE model (one descendent always has range size 1).

`max_minsize_as_function_of_ancsize`
If given, any state with a range larger that this value will be given a probability of zero (for the branch with the smaller rangesize). This means that not every possible combination of ranges has to be checked, which can get very slow for large state spaces.

`printmat` Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Details

The inputs `Rcpp_leftprobs` and `Rcpp_rightprobs` are basically irrelevant here, but retained for symmetry with the other functions. In effect, this function is identical with `rcpp_calc_anclikes_sp` except that `Rcpp_leftprobs` and `Rcpp_rightprobs` are arrays of 1s of `length(l)`, i.e. `length(number_of_states)`.

This function is no longer used in `BioGeoBEARS`, but has been retained to enable easy counting of the number of events. When all nonzero-probability events are of equal probability (e.g. as in LAGRANGE; Ree & Smith 2008) the function could be used for normalization, but it is safer to use `rcpp_calc_anclikes_sp` or one of the faster COO-like equivalents.

Value

`Rsp_rowsums` A vector of size (numstates) giving the number of events of nonzero probability for each ancestral states.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp, rcpp_calc_anclikes_sp_COOprobs, rcpp_calc_anclikes_sp_COOweights_faster
#bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013
@cite Matzke_2014 @cite ReeSmith2008

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

---

rcpp_calc_anclikes_sp_using_COOprobs

Calculate ancestral likelihoods given a COO-like probability matrix

---

Description

This function does a pass through a COO-like transition probability matrix for a node, inputting the probabilities that have been passed down from above for the left and right branch, and the sum of weights for each ancestral state, and returns the ancestral relative probabilities.

Usage

```
rcpp_calc_anclikes_sp_using_COOprobs(Rcpp_leftprobs, Rcpp_rightprobs,
  RCOO_left_i_list, RCOO_right_j_list, RCOO_probs_list, Rsp_rowsums,
  printmat = TRUE)
```

Arguments

Rcpp_leftprobs	Probabilities of the states at the base of the left descendant branch
Rcpp_rightprobs	Probabilities of the states at the base of the right descendant branch
RCOO_left_i_list	0-based index of the allowed left states
RCOO_right_j_list	0-based index of the allowed right states
RCOO_probs_list	Value of the specified nonzero cells

Rsp_rowsums	A vector of size (numstates) giving the sum of the relative probabilites of each combination of descendant states, assuming the probabilities of the left- and right-states are all equal (set to 1). This is thus the sum of the weights, and dividing by this normalization vector means that the each row of the speciation probability matrix will sum to 1. Default assumes the weights sum to 1 but this is not usually the case. Rsp_rowsums need only be calculated once per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.
printmat	Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Details

This C++ implementation should be slightly faster than the R version, although for a simple pass through an array the difference may not be great.

Value

R_anc_relprobs Vector of the probabilities of the ancestral states

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp

rcpp_calc_anclikes_sp #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

---

rcpp_calc_rowsums_for_COOweights_columnar

Calculate sum of weights for each ancestral state

Description

This is a C++ implementation of rcpp_calc_anclikes_sp_rowsums. It should be substantially faster, as it requires only one pass through COO_weights_columnar.

Usage

```
rcpp_calc_rowsums_for_COOweights_columnar(COO_weights_columnar,
  numstates = 1 + max(sapply(X = COO_weights_columnar, FUN = max)[1:3]),
  printmat = TRUE)
```

Arguments

COO_weights_columnar	Transition probability matrix in COO-like format as 4 columns: ancestral index, left index, right index, conditional probability given ancestral states. (assuming likelihood of descendants is 1). Indexes are 0-based. Keep in mind that cladogenesis matrices exclude the null state (a range of 0 areas), so if your states list starts with the null range (as is typical/default in DEC-style models) then to get the R 1-based state indices requires e.g. COO_weights_columnar[[1]] + 2.
numstates	The user should provide the number of states (WITHOUT counting the null range), in case they are not all present in COO_weights_columnar. If empty, the function assumes that the highest index represents the last state, and adds 1 to get the number of states. This may be a hazardous assumption.
printmat	Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Value

rowsums A vector of size (numstates) giving the sum of the relative probabilites of each combination of descendant states, assuming the probabilities of the left- and right-states are all equal (set to 1). This is thus the sum of the weights, and dividing by this normalization vector means that the each row of the speciation probability matrix will sum to 1. Default assumes the weights sum to 1 but this is not usually the case. Rsp_rowsums need only be calculated once per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

`rcpp_calc_splitlikes_using_COOweights_columnar` *Calculate the split likelihoods using* `COO_weights_columnar`

Description

Calculates the split likelihoods using `COO_weights_columnar`, i.e. the weights as produced by `rcpp_calc_anclikes_sp_COOweights_faster`.

Usage

```
rcpp_calc_splitlikes_using_COOweights_columnar(Rcpp_leftprobs,
  Rcpp_rightprobs, COO_weights_columnar, Rsp_rowsums, printmat = TRUE)
```

Arguments

`Rcpp_leftprobs`	Probabilities of the states at the base of the left descendant branch
`Rcpp_rightprobs`	Probabilities of the states at the base of the right descendant branch
`COO_weights_columnar`	Transition probability matrix in COO-like format as 4 columns: ancestral index, left index, right index, conditional probability given ancestral states. (assuming likelihood of descendants is 1). Indexes are 0-based. Keep in mind that cladogenesis matrices exclude the null state (a range of 0 areas), so if your states list starts with the null range (as is typical/default in DEC-style models) then to get the R 1-based state indices requires e.g. COO_weights_columnar[[1]] + 2.
`Rsp_rowsums`	A vector of size (numstates) giving the sum of the relative probabilites of each combination of descendant states, assuming the probabilities of the left- and right-states are all equal (set to 1). This is thus the sum of the weights, and dividing by this normalization vector means that the each row of the speciation probability matrix will sum to 1. Default assumes the weights sum to 1 but this is not usually the case. Rsp_rowsums need only be calculated once per tree+model combination, stored, and then re-used for each node in the tree, yielding significant time savings.
`printmat`	Should the probability matrix output be printed to screen? (useful for debugging, but can be dramatically slow in R.app for some reason for even moderate numbers of states; perhaps overrunning the line length...)

Value

`splitlikes` Vector of the probabilities of each allowed split

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014

Examples

```
# For the basic logic of a probablistic cladogenesis model, see
?rcpp_calc_anclikes_sp

# For examples of running the functions, see the comparison of all functions at:
# ?cladoRcpp
```

Rcpp_combn_zerostart *Get all the combinations of descendent state pairs, in 0-based index form*

Description

Given the number of states, this function returns all of the pairs of indexes corresponding to those states.

Usage

```
Rcpp_combn_zerostart(n_to_choose_from, k_to_choose, maxlim = 1e+07)
```

Arguments

n_to_choose_from	N in N choose K
k_to_choose	K in N choose K
maxlim	To avoid memory overruns, the number of combinations can be no larger than maxlim (default: 1e+07)

Details

The C++ version is MUCH faster than the plain-R version.

Value

outarray an integer matrix with outarray rows; the number of columns is the number of combinations.

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

rcpp_calc_anclikes_sp, rcpp_mult2probvect, rcpp_convolve #bibliography /Dropbox/_njm/__packages/cladoRcpp_s
@cite Matzke_2013 @cite Matzke_2014

Examples

```
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=2, maxlim=1e+07)
Rcpp_combn_zerostart(n_to_choose_from=4, k_to_choose=3, maxlim=1e+07)
```

---

rcpp_convolve *Run C++ version of convolve(x,y, conj=TRUE, type="open")*

Description

This function runs a C++ version of the R function convolve, specifically: `convolve(x,y, conj=TRUE, type="open")`

Usage

```
rcpp_convolve(a, b)
```

Arguments

a	a numeric vector
b	a numeric vector

Details

The R function convolve is an example of an R function that gets very slow when the input vectors are large. This C++ version, `rcpp_convolve` can be dramatically faster for large vectors.

`rcpp_convolve` produces the same output as: `convolve(ca, cb, conj=TRUE, type="open")`

Note: The C++ code is from the Rcpp examples in: Eddelbuettel & Francois (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, 40(8), 1-18.

Value

`convolve_result_vector` the vector which is the product of the convolution

Author(s)

C++ code by: Dirk Eddelbuettel <edd at debian.org> & Romain Francois (2011); This R wrapper & documentation: Nicholas Matzke `<matzke@berkeley.edu>`

See Also

Rcpp, convolve, rcpp_mult2probvect, Rcpp_combn_zerostart #bibliography /Dropbox/_njm/__packages/cladoRcpp_set
@cite Eddelbuettel_Francois_2011

Examples

```
# Set up 2 vectors, then convolve them
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
rcpp_convolve(a=ca, b=cb)

# Same as:
convolve(ca, cb, conj=TRUE, type="open")
```

`rcpp_mult2probvect` *Get the product of multiplying each pair of values in a vector (cross-product)*

Description

This function calls a C++ function which multiplies two vectors by each other elementwise, such that the output is of length(a) * length(b).

Usage

```
rcpp_mult2probvect(a, b)
```

Arguments

a	a numeric vector
b	a numeric vector

Details

This is the cross-product operation, which exists in R (`%o%` or `tcrossprod`). However, it is handy to have is as a C++ function for calculating the probability of pairs of descendant states, given the probability of each state individually.

Value

`tcross_product_vector` the vector which is the product of the convolution

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

See Also

`%o%`, `tcrossprod`, `Rcpp_combn_zerostart`, `rcpp_convolve`

Examples

```
ca = c(1,2,3,4,5)
cb = c(2,2,2,2,2)
rcpp_mult2probvect(a=ca, b=cb)

# Same as:
c(ca %o% cb)

# Or:
c(outer(ca, cb))

# Or:
tcrossprod(ca, cb)
```

`rcpp_states_list_to_DEmat` *C++ conversion of a states list to a dispersal-extinction matrix (DEmat)*

Description

This function takes a list of states/ranges, a matrix describing relative dispersal probability (dmat) for each pair of areas, and a list describing the local extirpation probability for each area (elist), and calculates a transition matrix Qmat accordingly.

Usage

```
rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist,
  amat = NULL, include_null_range = TRUE, normalize_TF = TRUE,
  makeCOO_TF = FALSE, min_precision = 1e-26)
```

Arguments

`areas_list`	a list of lists of areas (numbers, starting with 0)
`states_list`	a list of lists of areas (numbers, starting with 0)
`dmat`	dispersal matrix from area to area
`elist`	a list of extinction probabilities
`amat`	A matrix specifying the probability of instantaneous transition from one area to another (as in standard character rate matrices).
`include_null_range`	include the null () range (NA) in the matrix (LAGRANGE default=TRUE)
`normalize_TF`	should the columns be -1 * rowsums?
`makeCOO_TF`	should the returned matrix be COO or standard dense (the latter is default).
`min_precision`	what is the effective minimum size for 0

Details

The size of the matrix will expand dramatically with the number of areas. See numstates_from_numareas for the calculation.

Above 7 or so areas, making Qmat a COO-formatted matrix (COO=Coordinate list, see wikipedia, http://en.wikipedia.org/wiki/Sparse_matrix#Coordinate_list_.28COO.29) which can then be used in rexpokit's sparse-matrix algorithms, should be more efficient. (Sparse matrices are matrices made of mostly 0s.)

Value

dmat (a standard Q matrix)

Author(s)

Nicholas Matzke <matzke@berkeley.edu>

References

http://en.wikipedia.org/wiki/Sparse_matrix#Coordinate_list_.28COO.29 #bibliography /Dropbox/_njm/__packages/cladoRcpp_setup/cladoRcpp_refs.bib @cite Matzke_2013 @cite Matzke_2014 @cite ReeSmith2008

See Also

numstates_from_numareas, convolve

Examples

```
# Specify the areas
areas_list_txt = c("A", "B", "C")
areas_list_txt

# rcpp_states_list_to_DEmat function requires a 0-based list of areas
areas_list = seq(0, length(areas_list_txt)-1, 1)
areas_list

## Not run:

# Calculate the list of 0-based indices for each possible
#geographic range, i.e. each combination of areas
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=FALSE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=3,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=2,
include_null_range=TRUE)
states_list
states_list = rcpp_areas_list_to_states_list(areas=areas_list, maxareas=1,
include_null_range=TRUE)
```

```
states_list

# Hard-code the along-branch dispersal and extinction rates
d = 0.2
e = 0.1

# Calculate the dispersal weights matrix and the extinction weights matrix
# Equal dispersal in all directions (unconstrained)
areas = areas_list
distances_mat = matrix(1, nrow=length(areas), ncol=length(areas))
dmat = matrix(d, nrow=length(areas), ncol=length(areas))
dmat

# Equal extinction probability for all areas
elist = rep(e, length(areas))
elist

# Set up the instantaneous rate matrix (Q matrix, Qmat)
# DON'T force a sparse-style (COO-formatted) matrix here
force_sparse = FALSE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat

# DO force a sparse-style (COO-formatted) matrix here
force_sparse = TRUE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat


# Repeat with an amat
amat = dmat
amat[is.numeric(amat)] = 0.33

# Set up the instantaneous rate matrix (Q matrix, Qmat)
# DON'T force a sparse-style (COO-formatted) matrix here
force_sparse = FALSE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist, amat,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat

# DO force a sparse-style (COO-formatted) matrix here
force_sparse = TRUE
Qmat = rcpp_states_list_to_DEmat(areas_list, states_list, dmat, elist, amat,
include_null_range=TRUE, normalize_TF=TRUE, makeCOO_TF=force_sparse)
Qmat

## End(Not run)
```

strsplit3 *String splitting shortcut*

Description

strsplit returns the results inside a list, which is annoying. strsplit3 shortens the process.

Usage

```
strsplit3(x, ...)
```

Arguments

x	A string to split
...	Other arguments to strsplit. The argument split is *required*.

Value

out The output from inside the list.

Author(s)

Nicholas J. Matzke <matzke@berkeley.edu>

See Also

strsplit

Examples

```
# strsplit returns the results inside a list element
out = strsplit("ABC", split="")
out
# I.e....
out[[1]]

# If this is annoying/ugly in the code, use strsplit3:
out = strsplit3("ABC", split="")
out
```

Index