Package: phrapl (via r-universe)

August 22, 2024

Version 0.6.5

Date 2021-02-05

Title Phylogeography using Approximate Likelihood

Imports ape, binom, diagram, partitions, igraph, optimx, parallel, nloptr, Matrix, numDeriv, rgl, RColorBrewer, rgenoud, stats, utils, graphics, grDevices, phyclust

Suggests testthat (>= 2.1.0), animation, knitr, rmarkdown, covr

Description This creates and compares models for population structure within a species or between several related species, especially for questions in phylogeography. It can also help delimit populations into species. Models used can range from island models (populations exchanging genes through migration) to phylogeny models (populations splitting with no subsequent gene flow) to anything in between, such as subdivision with some gene flow. Input data are gene tree topologies. Models are fit using approximate likelihood: simulating gene tree evolution many times on the various population trees to estimate the likelihood of seeing the observed gene topologies.

Depends R (>= 3.5.0)

SystemRequirements perl

LazyData no

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.1.0

VignetteBuilder knitr

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

RemoteUrl https://github.com/bomeara/phrapl

RemoteRef HEAD

RemoteSha 318b236693bcfc8d763fdbde49bd10ecd2bc861f

Contents

phrapl-package . . . 2
AddEventToMigrationArray . . . 3
AddGrowthToAMigrationArray . . . 9
CalculateGdi . . . 9
CalculateModelAverages . . . 10
ConcatenateResults . . . 11
ConvertStructureDataToTrees . . . 13
GenerateMigrationArrayMap . . . 14
GenerateMigrationIndividuals . . . 15
GenerateMigrationIndividualsOneAtATime . . . 17
GetPermutationWeightsAcrossSubsamples . . . 20
GridSearch . . . 21
MergeTrees . . . 25
MsIndividualParameters . . . 26
phrapl-data . . . 27
phrapl-example . . . 28
PlotModel . . . 29
PlotModel2D . . . 30
PrepSubsampling . . . 32
PrepSubsamplingFiles . . . 33
ReadStructureData . . . 34
RunRaxml . . . 35
RunSeqConverter . . . 37
SaveMovie . . . 38

Index **39**

`phrapl-package` *Phylogeographic analysis using approximated likelihoods*

Description

phrapl compares phylogeographic models.

The complete list of functions can be displayed with `library(help = phrapl)`.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

AddEventToMigrationArray
Add an Event To a MigrationArray

Description

This function integrates a non-coalescence demographic event within a model or set of models. This can be useful if one wants to posit shifts in a parameter that do not correspond to a splitting event (e.g., one would like migration to only occur for a given length of time after a split, but then to cease).

Usage

```
AddEventToMigrationArray(migrationArray, eventTime, n0multiplierVec = NULL,
 growthVec = NULL, migrationMat = NULL)
```

Arguments

`migrationArray`	A list of models (or a single model) to which an event should be added
`eventTime`	The relative time period at which the new event should be inserted within the collapseMatrix
`n0multiplierVec`	A vector or single value specifying the n0multiplier parameter indices to be invoked during the new time interval. If `NULL`, no new set of parameter indices are invoked at this time.
`growthVec`	A vector or single value specifying the growth parameter indices to be invoked during the new time interval. If `NULL`, no new set of parameter indices are invoked at this time.
`migrationMat`	A matrix, vector, or single value specifying the migration parameter indices to be invoked during the new time interval. If `NULL`, no new set of parameter indices are invoked at this time.

Details

To use this function, one must specify a model (migrationIndividual) or set of models (migrationArray). If a set of models is specified, these models must all contain the same number of populations and the same number of collapse events (i.e., the collapseMatrix compenent within each migrationIndividual must have the same dimensions).

The relative timing of the desired new event must be specified as a single number using eventTime. An eventTime of 1 will place a new event (i.e., column) in the first column position within the collapseMatrix (and the other columns will be shifted to the right). An eventTime of 2 will place the new column in the second position, etc. The eventTime cannot exceed the number of events (i.e., columns) within the original collapseMatrix. The added column will consist entirely of NAs, indicating that no population coalescence can occur at this time.

Finally, one can specify a new set of n0multiplier, growth, and/or migration parameter indices to be invoked at the new specified time period using n0muliplierVec, growthVec, or migrationMat, respectively. When the default value for these is used (which is `NULL`), n0multiplier, growth, and migration matrices within each model are automatically expanded by simply copying over parameter indices from the adjacent time period to the new time period (i.e., no change is invoked). For n0multiplier and growth, a new column is added; for migration, a new matrix is added.

However, one can also specify the parameter indices to be used at this new time, allowing a shift in that parameter, without a correponding coalescence event. These indices can be specified in one of two ways:

First, one can simply specify a single parameter index which will be implemented for all relevant populations at that time period. For example, if the following collapseMatrix:

```
$collapseMatrix
      [,1] [,2]
[1,]     1    2
[2,]     1   NA
[3,]     0    2
```

is expanded to include a shift from non-zero migration among all populations to zero migration at some point prior to the coalescence of populations 1 and 2 (meaning that migration has occurred recently–upon secondary contact–but did not accompany initial divergence), we should set `eventTime = 1` and `migrationMat = 1`. Assuming that the original model included zero migration across all time periods, this will produce the following collpaseMatrix:

```
$collapseMatrix
     [,1] [,2] [,3]
[1,]   NA    1    2
[2,]   NA    1   NA
[3,]   NA    0    2
```

And the migrationArray will be expanded from two matrices:

```
$migrationArray
, , 1

     [,1] [,2] [,3]
[1,]   NA    0    0
[2,]    0   NA    0
[3,]    0    0   NA

, , 2

     [,1] [,2] [,3]
[1,]   NA   NA    0
[2,]   NA   NA   NA
[3,]    0   NA   NA
```

to three matrices:

```
$migrationArray
, , 1

     [,1] [,2] [,3]
[1,]   NA    1    1
[2,]    1   NA    1
[3,]    1    1   NA

, , 2

     [,1] [,2] [,3]
[1,]   NA    0    0
[2,]    0   NA    0
[3,]    0    0   NA

, , 3

     [,1] [,2] [,3]
[1,]   NA   NA    0
[2,]   NA   NA   NA
[3,]    0   NA   NA
```

where the new migration matrix was added in the first time period position and was filled with zeros, as per the value specified.

This approach works similarly for n0multiplier and growth. For example, if one would like to model population growth during the initial stages of divergence, followed by population size stability, one should set `eventTime = 1` and `growthVec = 0` to change this history:

```
$collapseMatrix
     [,1] [,2]
[1,]    1    2
[2,]    1   NA
[3,]    0    2

$growthMap
     [,1] [,2]
[1,]    1    0
[2,]    1   NA
[3,]    0    0
```

into this history:

```
$collapseMatrix
      [,1] [,2] [,3]
[1,]   NA    1    2
[2,]   NA    1   NA
[3,]   NA    0    2
```

```
$growthMap
      [,1] [,2] [,3]
[1,]    0    1    0
[2,]    0    1   NA
[3,]    0    0    0
```

If one wishes to specify more complex histories, in which different populations have different indices, one can alternatively specify the entire new vector (for n0multiplier and growth) or matrix (for migration) of parameter indices to be inserted. For example, in the previous example, if one would rather model population size stability upon divergence of populations 1 and 2 followed by recent growth of population 1, but not of populations 2 or 3, one can set `eventTime = 1` and `growthVec = c(1,0,0)` to change the growth history from this:

```
$growthMap
     [,1] [,2]
[1,]    0    0
[2,]    0   NA
[3,]    0    0
```

to this:

```
$growthMap
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    0   NA
[3,]    0    0    0
```

One can also specify a migration matrix using migrationMat. For example, if, one would like to insert a new matrix in which the migration rate differs among population pairs, one would specify

```
migrationMat = t(array(c(
  NA, 3, 2,
  1, NA, 1,
  2, 3, NA),
dim=c(3,3)))
```

If one would rather, this can also be specified simply as a vector (with or without the NAs). For example, the above migration matrix could be also specified as either `c(NA,1,2,3,NA,3,2,1,NA)` or as `c(1,2,3,3,2,1)`, where the vector is read first by column, then by row. Note that the number of values must match the number expected for that time period. For example, if one would like to insert a new migration history at ancestral time period 2, one should remember that there are fewer populations at that time period, and thus the number of values specified must be reduced accordingly. For example, specifying distinct migration parameters from population 1-2 to 3 versus from population 3 to 1-2 would be specified as either

```
migrationMat = t(array(c(
  NA, NA, 2,
  NA, NA, NA,
  1,  NA, NA),
dim=c(3,3)))
```

or as

```
migrationMat = (1,2)
```

Specifying two distinct population size parameters for ancestral population 1-2 and population 3 at the second time period would be done by setting `eventTime = 2` and `n0muliplierVec = c(1,2)`, to convert

```
$n0multiplierMap
     [,1] [,2]
[1,]    1    1
[2,]    1   NA
[3,]    1    1
```

into

```
$n0multiplierMap
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1   NA   NA
[3,]    1    2    1
```

Finally, note that although the function can only add a single time period to the collapseMatrix, one can add multiple non-coalescence time periods by repeatedly running the function on the same migrationArray.

Note

When analyzing a migrationArray using `GridSearch` that includes new time periods produced by `AddEventToMigrationArray`, a time value must be specified using the `addedEventTime` argument. `addedEventTime` takes either a single value or vector of values, such that there are as many values as added events in the collapseMatrix (sorted from most recent to most ancient). Time values can be specified as either a specific time period (in units of 4Ne) or as a relative time period. When the latter is desired, the `addedEventTimeAsScalar` argument in `GridSearch` must be set to `TRUE` (this is the default), which will cause the values specified by `addedEventTime` to be treated as scalars, rather than as absolute. Scalar values must be greater than zero and less than one, and will be multiplied by whatever the next collapse time estimate happens to be (which will typically vary across models). For example, setting `addedEventTimeAsScalar = TRUE` and `addedEventTime = 0.1` for the following model:

```
$collapseMatrix
      [,1] [,2] [,3]
[1,]   NA    1    2
[2,]   NA    1   NA
[3,]   NA    0    2

$growthMap
      [,1] [,2] [,3]
```

```
[1,]    0    1    0
[2,]    0    0   NA
[3,]    0    0    0
```

will cause (looking backwards in time) population growth to commence in population 1 after 10% of the branch length leading to coalescence of populations 1 and 2 has been reached.

One final thing to highlight is that the specified timing of these events must fit the modeled chronology of coalescence and non-coalescence events in the collapseMatrix. That is, the timing for an added non-coalescence event given by `addedEventTime` must be later than the collapse event directly preceding the added event and must be earlier than the collapse event directly following it. For this reason, it is recommended that `addedEventTimes` be entered as scalars, as this guarantees that the time will always be earlier than the collapse event that follows (but not that the time will occur after the previous collapse event). Absolute values can also be used, but remember that in a grid search, different values are explored, so many parameter combinations in the grid may not fit the specified event times. However, PHRAPL filters out all parameter combinations in the parameter grid that do not adhere to the specified chronology of events in the model. Thus, setting absolute dates may reduce the parameter space explored for a model, but this parameter space will always fit the model.

Author(s)

Nathan Jackson

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

Examples

```
# ##First, generate a migrationIndividual that models a
# ##specific coalescence history among four populations
# collapse_1<-c(0,0,1,1)
# collapse_2<-c(2,2,0,NA)
# collapse_3<-c(3,NA,3,NA)
# collapseList<-list(collapse_1,collapse_2,collapse_3)
# migrationIndividual<-GenerateMigrationIndividualsOneAtATime(collapseList=collapseList)
#
# ##Then, add a recent migration (secondary contact) event between sister populations 1 and 2
# migrationArray<-migrationIndividual
# eventTime=1
# migrationMat=c(1,0,0,1,0,0,0,0,0,0,0,0)
#
# migrationArray<-AddEventToMigrationArray(migrationArray=migrationArray,eventTime=eventTime,
#  migrationMat=migrationMat)
```

`AddGrowthToAMigrationArray` *Add growth paramter matrices (growthMaps) to a list of models (a migrationArray) that lack growth matrices*

Description

Previous versions of PHRAPL produced model lists (migrationArrays) that lack growth matrices (growthMaps). However, growthMaps must now be present to analyze models in PHRAPL, even if growth parameters are not incorporated in the model. Thus, to facilitate the analysis of old migrationArrays, this function takes as input a migrationArray lacking growthMaps and adds an empty growthMap to each model (i.e., a growthMap filled with zeros) within the migrationArray.

Usage

```
AddGrowthToAMigrationArray(migrationArray)
```

Arguments

`migrationArray`	A list of models to which NULL growth matrices should be added (i.e., matrices positing zero growth). A single model can also be specified.

Author(s)

Nathan Jackson

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

`CalculateGdi` *Calculate the genealogical divergence index (gdi)*

Description

This function calculates the genealogical divergence index (gdi) for a pair of taxa based on specified bi-directional migration rates (M) and a divergence time (t) estimated from a PHRAPL analysis. This index provides something similar to the genealogical sorting index (gsi), except for a given set of migration and divergence values instead of for a group on a tree. It also differs from the gsi in that it is calculated for a given pair of populations or taxa rather than for a single taxon in respect to the entire tree. The index is scaled to be between 0 and 1, where 0 = panmixia and 1 = strong divergence. Except when gdi is close to 1, there is a fairly high error, such that precision requires a large number of replicates (e.g., nreps = 10,000). We have incorporated the `binom.confint` function from the `binom` package to calculate a confidence interval if desired.

Usage

```
CalculateGdi(tau,migration.in,migration.out=NULL,nreps=10000,ciMethod="exact", msPath =
                    system.file("msdir", "ms", package = "P2C2M"))
```

Arguments

`tau`	The divergence time between two populations/groups.
`migration.in`	The migration rate into a population
`migration.out`	The migration rate out of a population. If this is not specified, migration rates are assumed to be symmetrical.
`nreps`	The number of genealogies to be simulated.
`ciMethod`	Which method to use to calculate a confidence interval for gdi. Any or all of eight methods for calculating a binomial confidence interval can be specified ("exact", "asymptotic", "agresti-coull", "wilson", "prop.test", "bayes", "logit", "cloglog", "probit", or "profile"). `ciMethod = "all"` will result in all eight being calculated. The default method is the Clopper-Pearson interval ("exact"). If `ciMethod=NULL`, only the gdi will be outputted.
`msPath`	Path to the local installation of ms; typing this string on the command line should result in ms running.

Author(s)

Nathan Jackson and Brian O'Meara

Examples

```
#CalculateGdi(tau=1,migration.in=1)
```

`CalculateModelAverages`

Calculate Model Averages

Description

This function outputs model averaged parameter estimates for a set of models.

Usage

```
CalculateModelAverages(totalData, averageAcrossAllModels = TRUE, parmStartCol = 9,
keep.na = FALSE)
```

Arguments

`totalData`	Path to input file (output of ConcatenateResults)
`averageAcrossAllModels`	If TRUE, each parameter is model averaged across all models in the dataset, even if that model does not include the given parameter. In these cases where a parameter is absent (i.e., the value is NA) this method simply assumes that the value for that parameter is zero. If FALSE, each parameter is model averaged by only considering those models that include the relevant parameter. The former method may be most appropriate for migration, as models that exclude this parameter are effectively setting migration to zero. One should be careful interpreting model averaged coalescence times (t), however. Keep in mind that when a model excluding a particular coalescent event recieves high support, this can be subject to different interpretations. For example, this could signal that these two populations are so similar that coalescence time between them is effectively zero. However, it could also signal that these two populations are very distinct, but that they coalesce with other populations prior to coalescing with each other.
`parmStartCol`	This gives the column number in totalData in which the parameter values begin. If using default PHRAPL output, this should be column 9.
`keep.na`	When TRUE, columns that only contain only NAs are retained in the output.

See Also

ConcatenateResults

Examples

```
#totalData<-load("/path/totalData.txt")
#modelAverages<-CalculateModelAverages(totalData)
```

`ConcatenateResults` *Concatenate PHRAPL Results from Different Runs*

Description

This function builds a table with details for each model analyzed using a grid or heuristic search, which can be spread across several phrapl result files. This table gives AICs, model weights, details of each model, and parameter estimates. The function can either import one or more R object files (with the extention .rda) located in the specified rdaFilePath, in which phrapl output is stored, or can be passed a vector of file names to be imported. This is the major output of phrapl.

Usage

```
ConcatenateResults(rdaFilesPath = "./", rdaFiles = NULL, migrationArray,
rm.n0 = TRUE, longNames = FALSE, addTime.elapsed = FALSE, addAICweights=TRUE,
outFile = NULL, nonparmCols=5)
```

Arguments

`rdaFilesPath`	Path to where R object files (which must have the extention .rda) are located which contain results from GridSearch (default is the current dir).
`rdaFiles`	As an alternative to reading in all .rda files in the specified directory, a vector of file names (located in the directory specified above) can be given.
`migrationArray`	The migrationArray used in the phrapl analysis. The model numbers in the output table will be matched to those in the migrationArray, thus it is important that these match up properly.
`rm.n0`	If TRUE, will exclude all n0multiplier parameters from the table. This is a way to de-clutter parameter output if models do not vary population size.
`longNames`	If TRUE, will use long descriptive names for each parameter rather than abbreviations
`addTime.elapsed`	If elapsed time is being tracked, this adds a column for this to the table.
`addAICweights`	If true, model ranks, differences in AIC (dAIC) and AIC weights (wAIC) are calculated and printed to the table.
`outFile`	If specified, gives the file name to which an output table will be printed.
`nonparmCols`	This gives the number of columns in the phrapl output that do not contain parameters.

Details

If subsets of models from a migrationArray are run in separate analyses, one should save the results as R object files (.rda) after each analysis (e.g., save(list=ls(), FILENAME)). Once all models are run, ConcatenateResults can then cull the results from all the files into a master result table.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

See Also

GridSearch, CalculateModelAverages.

Examples

```
## Not run:
totalData<-ConcatenateResults(
rdaFilesPath="/path_to_rdaFiles/",
rdaFiles=NULL,
outFile=NULL,
addAICweights=TRUE,
rmNaParameters=TRUE)

##Save results to a .txt file
write(t(totalData), file = "totalData.txt", ncolumns = 40, append = FALSE,
sep = "\t")

## End(Not run)
```

`ConvertStructureDataToTrees` *Use Structure data*

Description

This function uses the $snps element of the list returned by ReadStructureData() and creates a set of trees you can use in GridSearch or elsewhere with the doSNP=TRUE argument set. It returns a multiphylo object. Missing data (-9 in the original file) are removed.

Usage

```
ConvertStructureDataToTrees(snps)
```

Arguments

`snps`	The output of ReadStructureData, the snps object

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

Examples

```
#snptrees <- ConvertStructureDataToTrees(ReadStructureData(file="mydata")$snps)
```

`GenerateMigrationArrayMap`

Generate a MigrationArrayMap

Description

This function was meant to declare the matrix of model space search for a given migrationArray. It will generate a table which contains of all the parameter switches for each model. This is only a required input if estimating parameters using optimization rather than a grid search.

Usage

```
GenerateMigrationArrayMap(migrationArray)
```

Arguments

`migrationArray`	Contains a matrix of models to be explored.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

See Also

migrationArray

Examples

```
#load("/path/pkg/data/MigrationArray_3pop_3K.rda")
#GenerateMigrationArrayMap(migrationArray)
```

`GenerateMigrationIndividuals`

Generate a list of models (a migrationArray) to analyze using PHRAPL

Description

This function produces a list of all possible demographic models given a number of populations (specified by popVector), number of free parameters (specified by maxK), and other optional filtering criteria. The list of models is call a migrationArray (a name which, confusingly, is also given to migration matrices within a model) and each model is referred to as a migrationIndividual.

Each migrationIndividual is composed of four major demographic components: 1. The coalescence history (the collapseMatrix, in which rows correspond to populations and columns correspond to temporal events) 2. Population size scalar parameters (the n0multiplierMap, with the same dimensions as CollapseMatrix) 3. Growth (alpha) parameters (the growthMap, also with the same dimensions as CollapseMatrix) 4. Migration rate matrices (the migrationArray, where the number of matrices is equal to the number of temporal events specified by the collapseMatrix.

One can specify the total maximum number of free parameters (maxK), and also specify the maximum number of free parameters for a given parameter type (e.g., maxN0K, maxGrowthK, and maxMigrationK). Note that maxGrowthK and maxMigrationK can be set to zero; however the lowest possible value for N0K is one (in which case all population sizes are the same).

Specific demographic components can also be fixed, such that the generated migrationArray only varies subsets of parameters. Fixed components are specified by collapseList, n0multiplierList, growthList, and migrationList. Note that a collapseList must always be specified in order to specify any other demographic component. The format for specifying collapseList, n0multiplierList, and growthList are the same: as a list of vectors. A vector contains parameter values for each population (e.g., becoming the rows in collapseMatrix), and each vector in the list represents a different temporal event (e.g., becoming the columns in collapseMatrix). For example, collapseList = list(c(1,1,0),c(2,NA,2)) means that there are two coalescent events: in the first event, population 1 and 2 coalesce while population 3 does not; in the second event, ancestral population 1-2 coalesces with population 3.

MigrationList differs in that a list of matrices, rather than vectors, must be specified. There will be one migration matrix for the present generation, plus any historical matrices that apply (there will be as many matrices as there are collapse events). So, in the three population scenario, if there is symmetrical migration between populations 1 and 2 and no historical migration, migrationList will be:

migrationList<-list(
t(array(c(
NA, 1, 0,
1, NA, 0,
0, 0, NA),
dim=c(3,3))),

t(array(c(
NA, NA, 0,

NA, NA, NA,
0, NA, NA),
dim=c(3,3))))

Note that in R, arrays are constructed by reading in values from column 1 first then from column 2, etc. However, it is more intuitive to construct migration matrices by rows (i.e., first listing values for row 1, then row 2, etc). Thus, in the example above, arrays are entered as rows, and then transposed (using "t"). Also, spacing and hard returns can be used to visualize these values in the form of matrices.

Other methods of model filtering (using forceSymmetricalMigration or forceTree) can also be implemented.

Usage

```
GenerateMigrationIndividuals(popVector,maxK=SetMaxK(popVector),maxN0K=1,maxGrowthK=0,
 maxMigrationK=1,collapseList=NULL,n0multiplierList=NULL,growthList=NULL,
 migrationList=NULL, forceSymmetricalMigration=TRUE,forceTree=FALSE,
 verbose=FALSE,parallelRep=NULL)
```

Arguments

`popVector`	A vector that gives the number of samples for each population in a dataset. The model set generated by this function is based on the number of populations under consideration (indicated by length(popVector)).
`maxK`	The maximum number of free parameters to be incorporated into the model set. The default is calculated based on the number of samples in popVector.
`maxN0K`	The maximum number of n0 parameters to be incorporated into the model set.
`maxGrowthK`	The maximum number of growth parameters to be incorporated into the model set.
`maxMigrationK`	The maximum number of migration rate parameters to be incorporated into the model set.
`collapseList`	A particular collapse history to be specified
`n0multiplierList`	A particular n0muliplier history to be specified
`growthList`	A particular population growth history to be specified
`migrationList`	A particular migration history to be specified
`forceSymmetricalMigration`	If TRUE, migration rate indexes between populations are forced to be equal.
`forceTree`	If TRUE, only models with fully resolved topologies are included in the model set.
`verbose`	If TRUE, prints out status updates
`parallelRep`	If running the function through loop in parallel on a cluster, this argument is assigned a replicate number.

Note

For more information, please see the user manual.

Author(s)

Brian O'Meara and Nathan Jackson

Examples

```
# #Assuming a given tree and migration scenario, create a model set that tests all
# #possible distributions of a single population growth parameter across three populations.
#
# popVector<-c(3,3,3)
# maxK=5
# maxGrowthK=2
# forceTree=TRUE
#
# #Fix a tree
# collapse_1<-c(1,1,0)
# collapse_2<-c(2,NA,2)
# collapseList<-list(collapse_1,collapse_2)
#
# #Fix migration
# migration_1<-t(array(c(
#  NA, 1, 1,
#  1, NA, 1,
#  1, 1, NA),
# dim=c(3,3)))
#
# migration_2<-t(array(c(
#  NA, NA, 2,
#  NA, NA, NA,
#  2,  NA, NA),
# dim=c(3,3)))
# migrationList<-list(migration_1,migration_2)
#
# migrationArray<-GenerateMigrationIndividuals(popVector=popVector,maxK=maxK,maxGrowthK=maxGrowthK,
#  collapseList=collapseList,migrationList=migrationList,forceTree=forceTree)
```

`GenerateMigrationIndividualsOneAtATime`
Generate a single a priori PHRAPL model (MigrationIndividual)

Description

This function creates a single a priori model (migrationIndividual) for a given collapse, n0multiplier, growth, and migration history.

Usage

```
GenerateMigrationIndividualsOneAtATime(collapseList, n0multiplierList = NULL,
 growthList = NULL, migrationList = NULL)
```

Arguments

`collapseList`	List of collapse histories in the model
`n0multiplierList`	List of population size scalars in the model (if NULL, all are set to 1)
`growthList`	List of growth rate scalars in the model (if NULL, all are set to 0)
`migrationList`	List of migration rate matrices in the model (if NULL, all are set to 0)

Details

The four inputs for this function - collapseList, n0multiplierMap, growthMap, and migrationList - specify parameter indexes for the four major demographic components that can be modeled using PHRAPL.

CollapseList is the only set of parameters that must be specified. This gives a list of collapse history vectors, one vector for each coalescent event in the tree. So, collapseList = list(c(1,1,0),c(2,NA,2)) means that there are two coalescent events: in the first event, population 1 and 2 coalesce while population 3 does not; in the second event, ancestral population 1-2 coalesces with population 3.

The remaining three parameters may be specified or not specified. If not specified, (i.e., set to NULL),then null matrices will be automatically constructed in which all n0multipliers are set to one, and all growth and migration parameters are set to zero.

If specifying n0multiplierList and/or growthList, the format for these is the same as for collapseList, and the available parameters for these must match the splitting history depicted in the collapseList.

MigrationList is a list of migration matrices. There will be one migration matrix for the present generation, plus any historical matrices that apply (there will be as many matrices as there are collapse events). So, in the three population scenario, if there is symmetrical migration between populations 1 and 2 and no historical migration, migrationList will be:

migrationList<-list(
t(array(c(
NA, 1, 0,
1, NA, 0,
0, 0, NA),
dim=c(3,3))),

t(array(c(
NA, NA, 0,
NA, NA, NA,
0, NA, NA),
dim=c(3,3))))

Note that in R, arrays are constructed by reading in values from column 1 first then from column 2, etc. However, it is more intuitive to construct migration matrices by rows (i.e., first listing values for row 1, then row 2, etc). Thus, in the example above, arrays are entered as rows, and then transposed (using "t"). Also, spacing and hard returns can be used to visualize these values in the form of matrices.

Author(s)

Nathan Jackson and Brian O'Meara

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

Examples

```
# ##Four population example (with coalescence, growth, and migration parameters)
#
# #Create list of collapse histories
# collapse_1<-c(0,0,1,1)
# collapse_2<-c(2,2,0,NA)
# collapse_3<-c(3,NA,3,NA)
# collapseList<-list(collapse_1,collapse_2,collapse_3)
#
# #Create list of growth histories
# growth_1<-c(0,0,1,1)
# growth_2<-c(0,0,0,NA)
# growth_3<-c(0,NA,0,NA)
# growthList<-list(growth_1,growth_2,growth_3)
#
# #Create list of migration matrices
# migration_1<-t(array(c(
#  NA, 0, 0, 0,
#  0, NA, 0, 0,
#  0, 0, NA, 1,
#  0, 0, 1, NA),
# dim=c(4,4)))
#
# migration_2<-t(array(c(
#  NA, 0, 0, NA,
#  1, NA, 0, NA,
#  0, 0, NA, NA,
#  NA,NA,NA, NA),
# dim=c(4,4)))
#
# migration_3<-t(array(c(
#  NA, NA, 0, NA,
#  NA, NA, NA,NA,
#  0,  NA, NA,NA,
#  NA, NA, NA,NA),
# dim=c(4,4)))
#
# migrationList<-list(migration_1,migration_2,migration_3)
#
# #Run function
# migrationIndividual<-GenerateMigrationIndividualsOneAtATime(collapseList=collapseList,
# growthList=growthList, migrationList=migrationList)
# ##Three population Example (with coalescence and migration parameters)
#
# #Create list of collapse histories
# collapse_1<-c(1,1,0)
# collapse_2<-c(2,NA,2)
# collapseList<-list(collapse_1,collapse_2)
#
# #Create list of migration matrices
# migration_1<-t(array(c(
#  NA, 2, 1,
#  2, NA, 1,
#  1, 1, NA),
# dim=c(3,3)))
#
# migration_2<-t(array(c(
#  NA, NA, 3,
#  NA, NA, NA,
#  3,  NA, NA),
# dim=c(3,3)))
#
# migrationList<-list(migration_1,migration_2)
#
# #Run function
# migrationIndividual<-GenerateMigrationIndividualsOneAtATime(collapseList=collapseList,
#  migrationList=migrationList)
```

GetPermutationWeightsAcrossSubsamples

Calculate tip label degeneracy weights for a set of trees

Description

This function calculates weights for a tree or set of trees (typically subsampled trees) based on the degeneracy of tip lables (i.e., the proportion of times that permuting labels across a tree results in the same topology). These weights are applied during a grid or heuristic search in phrapl to obtain a more accurate estimate of the log likelihood of an observed tree given trees expected under a given model.

Usage

```
GetPermutationWeightsAcrossSubsamples(popAssignments,observedTrees)
```

Arguments

popAssignments	A list of vectors (typically only one vector will be specified) that define the number of tips per population included in the observed trees (usually these will be subsampled trees).

`observedTrees`	Multiphylo object containing the subsampled trees (outputted from PrepSubsampling)

Note

Because this function exhaustively permutes all possible tip lables across a tree, it can take a long time for large subsampled trees. For more information, please see the user manual.

Note

For more information, please see the user manual.

Author(s)

Nathan Jackson & Brian O'Meara

`GridSearch` *Computes AIC for a given model or models on a fixed set of parameters*

Description

This function takes a given model or models and a grid of parameters and returns the AIC at each parameter. In testing, it was found to work better than traditional optimization approaches. However, it can still do a heuristic search.

Usage

```
GridSearch(modelRange = c(1:length(migrationArray)), migrationArrayMap=NULL,
migrationArray, popAssignments, badAIC = 1e+14,
maxParameterValue = 20, nTrees = 1e5,
msPath = system.file("msdir", "ms", package = "P2C2M"),
comparePath = system.file("extdata","comparecladespipe.pl", package = "phrapl"),
observedTrees, subsampleWeights.df = NULL, doWeights = TRUE,
unresolvedTest = TRUE, print.ms.string = FALSE, print.results = TRUE,
print.matches = FALSE, debug = FALSE, method = "nlminb", itnmax = NULL,
ncores = 1, results.file = NULL, maxtime = 0, maxeval = 0, return.all = TRUE,
numReps = 0, startGrid = NULL,
collapseStarts = c(0.3, 0.58, 1.11, 2.12, 4.07, 7.81, 15),
n0Starts = c(0.1, 0.5, 1, 2, 4),
growthStarts=c(0.30,0.58,1.11,2.12,4.07,7.81,15.00),
migrationStarts = c(0.1, 0.22, 0.46, 1, 2.15, 4.64),
subsamplesPerGene = 1, totalPopVector = NULL, summaryFn = "mean",
saveNoExtrap = FALSE, doSNPs = FALSE, nEq=100, setCollapseZero=NULL,
dAIC.cutoff=2, rm.n0 = TRUE, popScaling = NULL, checkpointFile = NULL,
parameter.ambiguous = FALSE, addedEventTime = NULL, addedEventTimeAsScalar = TRUE,
optimization = "grid", genoudPopSize = 25, numGridStartVals = 25,
solutionTolerance = 1, skipGrid = FALSE, usePhyclust=TRUE, ...)
```

Arguments

`modelRange`	Integer vector: which models to examine. Do not specify to use default of all models.
`migrationArrayMap`	A data.frame containing information about all the models. Only required for heuristic search, not for grid search.
`migrationArray`	List containing all the models
`popAssignments`	A list of vectors (typically only one vector will be specified) that define the number of individuals per population included in the observed tree file (usually these will be subsampled trees). Defining popAssignments as list(c(4,4,4)) for example means that there 12 tips per observed tree, with 4 tips per population.
`badAIC`	In case of failure (such as trying a parameter outside a bound), this allows returning of suboptimal but still finite number. Mostly used for heuristic searches.
`maxParameterValue`	A bound for the maximum value for any parameter.
`nTrees`	Integer: the number of trees to simulate in ms.
`msPath`	Path to the local installation of ms; typing this string on the command line should result in ms running.
`comparePath`	Path to the local placement of the compareCladesPipe.pl perl script, including that script name.
`observedTrees`	Multiphylo object of the empirical trees.
`subsampleWeights.df`	A dataframe of the weights for each subsample. If this is NULL, it is computed within GridSearch.
`doWeights`	In no subsampleWeights.df object is called and doWeights = TRUE, subsample weights will be calculated for each tree prior to AIC calculation.
`unresolvedTest`	Boolean: deal with unresolved gene trees by looking for partial matches and correcting for that.
`print.ms.string`	Mostly for debugging, Boolean on whether to verbosely print out the calls to ms.
`print.results`	If TRUE, after each simulation cycle, parameters and AICs are printed to the screen.
`print.matches`	Mostly for debugging, Boolean on whether to verbosely print out the matches.
`debug`	Whether to print out additional debugging information.
`method`	For heuristic searches, which method to use. ?optim for more information.
`itnmax`	For heuristic searches, how many steps.
`ncores`	Allows running on multiple cores. Not implemented yet.
`results.file`	File name for storage of results.
`maxtime`	Maximum run time for heuristic search.
`maxeval`	Maximum number of function evaluations to run for heuristic search.

`return.all`	Boolean: return just the AIC scores or additional information.
`numReps`	For heuristic searches, number of starting points to try.
`startGrid`	Starting grid of parameters to try. Leave NULL to let program create this.
`collapseStarts`	Vector of starting values for collapse parameters.
`n0Starts`	Vector of starting values for n0.
`growthStarts`	Vector of starting values for growth parameters.
`migrationStarts`	Vector of starting values for migration rates.
`subsamplesPerGene`	How many subsamples to take per gene
`totalPopVector`	Overall number of samples in each population before subsampling.
`summaryFn`	Way to summarize results across subsamples.
`saveNoExtrap`	Boolean to tell whether to save extrapolated values. FALSE by default.
`doSNPs`	Boolean to tell whether to use the SNPs model: count a single matching edge on the simulated tree as a full match. FALSE by default.
`nEq`	If no simulated trees match the observed trees, the frequentist estimate of the matching proportion is exactly zero (flip a coin 5 times, see no heads, so estimate probability of heads is zero). This would have an extreme effect: if any gene trees don't match, the model has no likelihood. A better approach is to realize that a finite set of samples gives finite information. We assume that the probability of a match, absent data, is 1/number of possible gene trees, and this is combined with the empirical estimate to give an estimate of the likelihood. A question is how much weight to put on this pre-existing estimate, and that is set by nEq. With its default of 100, very low weight is placed on this: it's equivalent in the info present in nTrees=100, and since the actual nTrees is 10000 or more, it has very little impact.
`setCollapseZero`	A vector of collapse parameters that will be set to zero (e.g., c(1,2) will set both the first and second collapse parameters to zero). K will be adjusted automatically to account for the specified fixed parameters.
`dAIC.cutoff`	A value specifying how optimized parameter values should be selected. Parameter estimates are calculated by taking the mean parameter value across all values within an AIC distance of dAIC.cutoff relative to the lowest AIC value. The default is 2 AIC points. Note that this argument only applies when calculating parameters with the ambiguous name format that was used prior to December 2015.
`rm.n0`	A boolean indicating whether n0multiplier parameter estimates should be outputted with the other estimates, even if the value is always the same. This declutters output if one is not analyzing models that vary population sizes. Note that phrapl checks whether there are multiple n0multiplier values in the inputted migrationArray; if there are, rm.n0 is automatically switched to be FALSE.
`popScaling`	A vector whose length is equal to the number of loci analyzed that gives the relative scaling of effective population size for each locus (e.g., diploid nuclear locus = 1, X-linked locus = 0.75, mtDNA locus = 0.25). Default is equal scaling for all loci.

`checkpointFile`	Results can be printed to a file which acts as a checkpoint. If a job is stopped during a GridSearch, rather than starting the analysis over, if a file was previously specified using this argument, the search will resume at the most recent iteration printed to the file. Currently, this argument can only be used when running a single model in GridSearch. However, if GridSearch is taking a long time when running multiple models at once, one can save results more often by reducing the number of models run at a time. Thus, the checkpointFile argument is only really necessary when you've got a single model running that can't be broken up any further.
`parameter.ambiguous`	If true, calculates parameters with the ambiguous name format that was used prior to December 2015, rather than using the unambiguous parameter names format.
`addedEventTime`	This takes either a single value or vector of values giving absolute or relative times at which non-coalescence events that have been added to a migrationArray occur (such events are added using the function `AddEventToMigrationArray`). There should be as many values listed as there are added events in the collapseMatrix (sorted from most recent to most ancient). Time values can be specifed as either an absolute time period (in units of 4Ne) or as a relative time period. When the latter is desired, the `addedEventTimeAsScalar` argument (see below) must be set to `TRUE`, which will cause the values specified by `addedEventTime` to be treated as scalars, rather than as absolute. Scalar values must be greater than zero and less than one, and will be multiplied by whatever the next collapse time estimate happens to be (which will typically vary across models and across a search grid). Note that the specified timing of these events must fit the modeled chronology of coalescent and non-coalescence events in the collapseMatrix. That is, the timing for an added non-coalescence event given by `addedEventTime` must be later than the collapse event directly preceding the added event and must be earlier than the collapse event directly following it. For this reason, it is recommended that `addedEventTimes` be entered as scalars, as this guarantees that the time will always be earlier than the collapse event that follows (but not that the time will occur after the previous collapse event). Absolute values can also be used, but remember that in a grid search, different values are explored, so many parameter combinations in the grid may not fit the specified event times. However, PHRAPL filters out all parameter combinations in the parameter grid that do not adhere to the specified chronology of events in the model. Thus, setting absolute dates may reduce the parameter space explored for a model, but this parameter space will always fit the model.
`addedEventTimeAsScalar`	Dictates whether added event times set by addedEventTime are treated as scalars (when `TRUE`) or as absolute values (when `FALSE`).
`optimization`	Dictates how parameters are estimated. Options are `"grid"`, `"nloptr"`, or `"genoud"`. If `"nloptr"` or `"genoud"` are selected, this causes PHRAPL to first run a grid search to obtain starting values; then further optimization occurs using an algorithm.
`genoudPopSize`	Specifies the population size (pop.size) value used by genoud.
`numGridStartVals`	Specifies the number of grid value combinations used as starting values for

	genoud (where grid values are sorted by AIC such that the best models are preferentially chosen). If one carries out an initial grid search to obtain starting values for genoud (`skipGrid = FALSE`), then smaller `numGridStartVals` values will give more weight to the best values inferred from a grid search. Using a large number for `numGridStartVals` will dampen the impact of grid search optimization. Note that if `numGridStartVals` is less than `genoudPopSize`, genoud will automatically add in addition "individuals". If `numGridStartVals` exceeds the number of grid values for a given model, genoud will simply add more individuals such that `genoudPopSize` is satisfied.
`solutionTolerance`	This gives the `solution.tolerance` level used by genoud
`skipGrid`	If set to `TRUE`, the initial grid search is skipped when doing genoud optimization. In this case, starting values are determined by randomly selecting values from the grid (the number of which is determined by `numGridStartVals`).
`usePhyclust`	If TRUE, use phyclust's version of ms rather than P2C2M (should be equivalent)
`...`	Other items to pass to heuristic search functions.

Details

We recommend using the grid, not the heuristic search. If you are using the SNPs model, you should have uniform weights for the gene trees unless there is some reason you want to weight them differently.

Value

If return.all==FALSE, just a vector of AIC values. Otherwise, a list with parameters used for the grid and AIC for each, if using grid search.

Note

For more information, please see the user manual.

Author(s)

Brian O'Meara & Nathan Jackson

`MergeTrees` *Merge Trees*

Description

This function merges all the *.tre files in a directory into a single file called "trees.tre". The files must be in newick format. These tree files must have the same label for individuals across all loci. These labels must be the same listed in assignFile ("cladeAssignments.txt").

Usage

```
MergeTrees(treesPath)
```

Arguments

`treesPath`	Path to tree files in newick format.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

Examples

```
#MergeTrees(treePath="/path_to_tree_files/")
```

`MsIndividualParameters` *Show the Parameters within a given PHRAPL model (migrationIndividual)*

Description

Show the distribution of parameters of a certain model for a given migrationArray. The considered parameters are: coalescent events (collapse), population size (n0multiplier) or, different migration parameters (migration).

Usage

```
MsIndividualParameters(migrationIndividual)
```

Arguments

`migrationIndividual`	This is the chosen model. If you want model 17 from migrationArray, you could pass this in as migrationIndividual=migrationArray[[17]]

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

See Also

migrationArray

Examples

```
## Get the parameters in model 6

data(MigrationArray_3pops_maxMigrationK1)
MsIndividualParameters(migrationArray[[6]])
```

`phrapl-data` *Cached migrationArrays*

Description

These are precomputed migrationArrays that have been generated using GenerateMigrationIndividuals according to specific critera. All the model sets avaiable below contain only models with two type of parameters: coalescence times and migration rates (i.e., equal and stable population sizes are invoked in all models) and fully resolved trees (i.e., forceTree = TRUE).

To load a migrationArray with 3 populations, a total maximum of three parameters (maxK = 3), a maximum of one migration rate (MaxMigrationK = 1), and the possibility of assymetrical migration (forceSymmetricalMigration = FALSE), type

`data(MigrationArray_3pops_maxMigrationK1_asymmetrical)`.

To load a migrationArray with three populations, a total maximum of three parameters (maxK = 3), a maximum of one migration rate (MaxMigrationK = 1), and only symmetrical migration (forceSymmetricalMigration = TRUE), type

`data(MigrationArray_3pops_maxMigrationK1)`.

To load a migrationArray with three populations, a total maximum of four parameters (maxK = 4), a maximum of two migration rates (MaxMigrationK = 2), and the possibility of assymetrical migration (forceSymmetricalMigration = FALSE), type

`data(MigrationArray_3pops_maxMigrationK2_asymmetrical)`.

To load a migrationArray with three populations, a total maximum of four parameters (maxK = 4), a maximum of two migration rates (MaxMigrationK = 2), and only symmetrical migration (forceSymmetricalMigration = TRUE), type

`data(MigrationArray_3pops_maxMigrationK2)`.

To load a migrationArray with three populations, a total maximum of five parameters (maxK = 5), a maximum of three migration rates (MaxMigrationK = 3), and the possibility of assymetrical migration (forceSymmetricalMigration = FALSE), type

`data(MigrationArray_3pops_maxMigrationK3_asymmetrical)`.

To load a migrationArray with three populations, a total maximum of five parameters (maxK = 5), a maximum of three migration rates (MaxMigrationK = 3), and only symmetrical migration (forceSymmetricalMigration = TRUE), type

`data(MigrationArray_3pops_maxMigrationK3)`.

To load a migrationArray with three populations, a total maximum of six parameters (maxK = 6), a maximum of four migration rates (MaxMigrationK = 4), and the possibility of assymetrical migration (forceSymmetricalMigration = FALSE), type `data(MigrationArray_3pops_maxMigrationK4_asymmetrical).`

To load a migrationArray with four populations, a total maximum of four parameters (maxK = 4), a maximum of one migration rate (MaxMigrationK = 1), and only symmetrical migration (forceSymmetricalMigration = TRUE), type

`data(MigrationArray_4pops_maxMigrationK1).`

To load a migrationArray with four populations, a total maximum of five parameters (maxK = 5), a maximum of two migration rates (MaxMigrationK = 2), and only symmetrical migration (forceSymmetricalMigration = TRUE), type

`data(MigrationArray_4pops_maxMigrationK2).`

Examples

```
#Load migrationArray
data(MigrationArray_3pops_maxMigrationK1_asymmetrical)
#Inspect the number of models
length(migrationArray)
```

`phrapl-example` *A description for applying phrapl to a test dataset*

Description

A test dataset called TestData.rda can be found in the data directory included in the phrapl package. This dataset comprises 10 gene trees containing three populations ("A","B", and "C"), with 20 samples per population, and a single outgroup sample ("D"). Genealogies were simulated under a coalescent isolation only model where the true history is where populations A and B coalesce after 4Ne generations, population A-B coalesces with C after 10Ne generations and population A-B-C coalesces with D after 20Ne generations.

To access this dataset, simply load the file into R; e.g., data(TestData).

Five objects are included: 1. migrationArray, which lists specifications for six models to be analyzed (model number 1 corresponds to the correct model); 2. migrationArrayMap, which lists parameter indexes for each model included in migrationArray; 3. trees, which contains the 10 full gene trees in newick format; 4. assignFile, which gives assignments of samples to populations; 5. popAssignments, which gives a list of vectors describing how the dataset should be subsampled (e.g., list(c(3,3,3)) means that one subsampling procedure should be carried out, where 3 individuals are sampled from populations A, B, and C).

Subsampling (using PrepSubsampling) and weights calculation (using GetPermutationWeightsAcrossSubsamples) can first be carried out using assignFile popAssignments, and trees.

A phrapl GridSearch can be run using the output from these functions in addition to migrationArray and migrationArrayMap.

Examples

```
data(TestData)
## Not run:
#Run PrepSubsampling on this dataset to get
#10 subsample iterations per tree. Include
#everything in the working directory
observedTrees<-PrepSubsampling(assignmentsGlobal=assignFile,
  observedTrees<-trees,popAssignments=popAssignments,
  subsamplesPerGene=10)

#To get the weights
subsampleWeights.df<-GetPermutationWeightsAcrossSubsamples(
  popAssignments=popAssignments,observedTrees=observedTrees)

#To get migrationArrayMap
migrationArrayMap <- GenerateMigrationArrayMap(migrationArray)

#To run a grid search
result<-GridSearch(migrationArrayMap=migrationArrayMap,migrationArray=
  migrationArray,modelRange=1:6,popAssignments=popAssignments,
  observedTrees=observedTrees,subsampleWeights.df=subsampleWeights.df,
  subsamplesPerGene=10,totalPopVector=rep(20,10),nTrees=1e3,
  print.results=TRUE,return.all=TRUE)

#To save the results to a file
save(list=ls(), file="phraplOutput.rda")

#To generate an output table
totalData<-ConcatenateResults(rdaFiles="phraplOutput.rda",
 outFile="AIC_table.txt")

## End(Not run)
```

PlotModel *Create a 3D spinning plot for a given model*

Description

This uses rgl to create a depiction of a given model, complete with migration arrows. You can spin this using the cursor.

Usage

```
PlotModel(migrationIndividual, parameterVector = NULL,taxonNames = NULL,
  time.axis = FALSE, time.axis.col ="black", apply.base = FALSE,
  base.color = "black", new.window = TRUE, base.bounds = 2)
```

Arguments

`migrationIndividual`	This is the chosen model. If you want model 17 from migrationArray, you could pass this in as migrationIndividual=migrationArray[[17]]
`parameterVector`	If you have numeric parameter estimates, you can pass them in here. This affects things like node height, migration rate arrow width, etc. Otherwise, it will plot a generic model.
`taxonNames`	If instead of population A, B, ... you want Oahu, Hawaii, ... you can pass them in as a vector.
`time.axis`	If you want to have a time axis, enter TRUE.
`time.axis.col`	If you want a particular color for the time axis, enter it here.
`apply.base`	This will draw a box at the rootward end of the population history. This can be convenient for 3D printing.
`base.color`	What color should the box be, if one is drawn.
`new.window`	If FALSE, overwrites the current image if one exists. By default, TRUE, so each time you run this function you create another window.
`base.bounds`	This changes the size of the base.

Details

This requires rgl, which periodically becomes hard to install on certain platforms.

Value

Creates a stunning image.

Author(s)

Brian O'Meara

See Also

`SaveMovie`, ~~~

`PlotModel2D` *Create a 2D plot for a given model*

Description

This creates a two-dimensional model

Usage

```
PlotModel2D(migrationIndividual, parameterVector=NULL, taxonNames=NULL,
 pad.for.input=FALSE, tree.lwd=30, tree.col="black", arrow.lwd=1,
 arrow.col="red", arrow.lty="solid", tip.cex = 1, tip.col = "black")
```

Arguments

`migrationIndividual`	This is the chosen model. If you want model 17 from migrationArray, you could pass this in as migrationIndividual=migrationArray[[17]]
`parameterVector`	If you have numeric parameter estimates, you can pass them in here. This affects things like node height, migration rate arrow width, etc. Otherwise, it will plot a generic model.
`taxonNames`	If instead of population A, B, ... you want Oahu, Hawaii, ... you can pass them in as a vector.
`pad.for.input`	[not implemented yet]
`tree.lwd`	Line width for the tree
`tree.col`	Color for the tree
`arrow.lwd`	Line width for migration arrows
`arrow.col`	Color for migration arrows
`arrow.lty`	Line type for migration arrows
`tip.cex`	Specify tip size
`tip.col`	Specify tip color

Details

This will create a plot showing migration, population collapse, and other features. It does not yet use parameter estimates. It shows a gradient for branches at the base of the tree to represent the history being lost in the mists of time.

Value

Creates a stunning image.

Author(s)

Brian O'Meara

See Also

`SaveMovie`, ~~~

Examples

```
data(MigrationArray_3pops_maxMigrationK1)
pretty.model <- migrationArray[[1]]
PlotModel2D(pretty.model)
```

`PrepSubsampling` *Subsample tips from a tree*

Description

This function subsamples tips from a tree according to a specified sampling strategy. Subsampling is ordinarily done by specifying a fixed number of individuals to take per population (using popAssignments). An outgroup population can be included to root each subsample, but then pruned from the tree prior to analysis using phrapl.

Usage

```
PrepSubsampling(assignmentsGlobal, observedTrees,popAssignments,
subsamplesPerGene,nIndividualsDesired=NULL,minPerPop=1, outgroup=TRUE,
outgroupPrune=TRUE)
```

Arguments

`assignmentsGlobal`	Dataframe with population assignments. This should consist of two columns: the first lists all individuals in the tree; the second lists populations to which individuals belong (e.g., "A", "B", "C"...).
`observedTrees`	A multiphylo object with observed trees.
`popAssignments`	A list of vectors that gives a fixed number of subsamples to take per population. Defining popAssignments as list(c(4,4,4)) for example will result in 4 subsamples being taken from each of three populations. The number of indivduals to subsample can vary among populations. If more than one subsampling vector is included,all subsampling iterations will first be carried out according to the first vector, then subsequent vectors, for a given input tree.
`subsamplesPerGene`	Gives the number of subsampling iterations per tree.
`nIndividualsDesired`	The total number of individuals to be included in a subsample. This must only be specified if one wishes to randomly subsample tips by only fixing the total number to be subsampled, but not fixing the specific number to be taken per population (which is done by specifying popAssignments). This option must be specified in conjuction with minPerPop.
`minPerPop`	If one wishes to subsample from a tree, but not fix the number of samples to be taken per population, this gives the minimum number of individuals to include from any population in a subsample. This is only used in conjunction with nIndividualsDesired. If popAssignments is defined, minPerPop will be ignored.
`outgroup`	Whether an outgroup is included in the tree to be used to root the subsampled trees. If TRUE, one sample from the outgroup population will be included in each subsampled tree. The outgroup population is identified as the last population included in the assignFile.

`outgroupPrune`	Whether a subsampled outgroup should be pruned from the subsampled tree prior to being printed to the output file. This is to be used in conjunction with the outgroup option.

Note

For more information, please see the user manual.

Author(s)

Nathan Jackson & Brian O'Meara

`PrepSubsamplingFiles` *Subsample tips from a tree*

Description

This function subsamples tips from a tree according to a specified sampling strategy. Subsampling is ordinarily done by specifying a fixed number of individuals to take per population (using popAssignments). An outgroup population can be included to root each subsample, but then pruned from the tree prior to analysis using phrapl.

Usage

```
PrepSubsamplingFiles(subsamplePath="./",assignFile="cladeAssignments.txt",
treesFile="trees.tre", outputFile="observed.tre",popAssignments,
subsamplesPerGene,nIndividualsDesired=NULL,minPerPop=1,outgroup=TRUE,
outgroupPrune=TRUE)
```

Arguments

`subsamplePath`	Path to the directory from which input files are accessed and to which output files are printed.
`assignFile`	File that includes population assignments. This should consist of two columns: the first lists all individuals in the tree; the second lists populations to which individuals belong (e.g., "A", "B", "C"...). The file must include a header row.
`treesFile`	A file that includes one or more trees to be subsampled, in newick format.
`outputFile`	File to which subsampled trees will be printed. If there is more than one tree in the treesFile, all subsamples for the first tree will be printed, then all the subsamples for the second tree, etc.
`popAssignments`	A list of vectors that gives a fixed number of subsamples to take per population. Defining popAssignments as list(c(4,4,4)) for example will result in 4 subsamples being taken from each of three populations. The number of indivduals to subsample can vary among populations. If more than one subsampling vector is included,all subsampling iterations will first be carried out according to the first vector, then subsequent vectors, for a given input tree.

`subsamplesPerGene`
: Gives the number of subsampling iterations per tree.

`nIndividualsDesired`
: The total number of individuals to be included in a subsample. This must only be specified if one wishes to randomly subsample tips by only fixing the total number to be subsampled, but not fixing the specific number to be taken per population (which is done by specifying popAssignments). This option must be specified in conjuction with minPerPop.

`minPerPop`
: If one wishes to subsample from a tree, but not fix the number of samples to be taken per population, this gives the minimum number of individuals to include from any population in a subsample. This is only used in conjunction with nIndividualsDesired. If popAssignments is defined, minPerPop will be ignored.

`outgroup`
: Whether an outgroup is included in the tree to be used to root the subsampled trees. If TRUE, one sample from the outgroup population will be included in each subsampled tree. The outgroup population is identified as the last population included in the assignFile.

`outgroupPrune`
: Whether a subsampled outgroup should be pruned from the subsampled tree prior to being printed to the output file. This is to be used in conjunction with the outgroup option.

Note

For more information, please see the user manual.

Author(s)

Nathan Jackson & Brian O'Meara

`ReadStructureData` *Read Structure data*

Description

This function reads a Structure file for SNPs. It assumes you have states 1 and 2 and that missing data is encoded as -9.

It returns a list with three objects:

$snps the data to pass into ConvertStructureDataToTrees

$sample.names the first column of sample names

$populations the second column in the original data, assumed (but not required) to be population names.

Usage

```
ReadStructureData(file, pairs)
```

Arguments

`file`	A file with the structure data
`pairs`	If 0, take all columns and rows. If 1, assume individuals are repeated on rows and so you should sample one row at random per site. If 2, assume individuals are represented as duplicated columns, so choose samples at random.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

Examples

```
#snptrees <- ConvertStructureDataToTrees(ReadStructureData(file="mydata")$snps)
```

`RunRaxml` *Run Raxml on a set of trees*

Description

This function runs RAxML through phrapl. The function will produce an input string for RAxML and calling it up. RAxML must be installed in the system before invoking this function. It reads in all phylip files in the designated path, and runs RAxML for each in turn. Outgroups and mutation models can be specified either as a single string to be used for all loci or as a vector which needs to match the order of the reading of the phylip files (i.e., alphabetic/numeric).

Usage

```
RunRaxml(raxmlPath = paste(getwd(),"/",sep=""), raxmlVersion = "raxmlHPC",
inputPath = paste(getwd(),"/",sep=""), mutationModel, outgroup, iterations,
seed = sample(1:1e+07, 1), outputSeeds = FALSE, discard = FALSE)
```

Arguments

`raxmlPath`	Path to the RAxML executable (default is current dir).
`raxmlVersion`	Name and version of the executable (default is "raxmlHPC").
`inputPath`	Path to sequence files in phylip format.
`mutationModel`	Mutation model (e.g., "GTRGAMMA"). If there is only one, it can be simply given as a single string. If there is a different model for each locus, the path to a text file can be provided that includes a single column of models that is the same length as the number of sequence files (note that the ordering must match the order of the phylip files in the directory (i.e., alphabetic/numeric)).

`outgroup`	A character string denoting an outgroup. Like with mutation models, a single outgroup can be given to use for all sequence files, or the location of a text file can be given which provides a vector (a single column) with the outgroups.
`iterations`	The number of iterations of the search algorithm.
`seed`	Provide a seed (a number). The default is a random number
`outputSeeds`	If "outputSeeds=TRUE", a file is saved that includes the seeds used for each dataset
`discard`	If "discard=TRUE", all RAxML output except the most likely tree is automatically discarded

Details

For more information see http://bodegaphylo.wikispot.org/RAxML_Tutorial.

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

See Also

RunSeqConverter

Examples

```
#raxmlPath<-load("/path_to_RAxML_executable/")
#inputPath<-load("/path/sequence_files.phylip")

#RunRaxml(raxmlPath=raxmlPath,
#raxmlVersion="raxmlHPC",
#inputPath=inputPath,
#mutationModel="GTRGAMMA",
#outgroup="outgroup",
#iterations=2,
#seed=sample(1:10000000,1),
#outputSeeds=FALSE,
#discard=FALSE)

## The output file will be located in your current dir and will be named
##"RAxML_bestTree.YOUR_FILE_NAME.phylip.tre"
```

RunSeqConverter — *Run SeqConverter to convert across sequence formats*

Description

This function runs a perl script that takes one or more sequence files in nexus, fasta, or Se-Al formats and converts them to (relax) phylip format. This format is required for RAxML (RunRaxml).

Usage

```
RunSeqConverter(seqConvPath=system.file("extdata","seqConverter.pl",package="phrapl"),
 inFilePath="./",inputFormat="nex")
```

Arguments

`seqConvPath`	Path to the seqConverter script (default is to the script distributed with phrapl).
`inFilePath`	Path to the sequence files (default is current dir).
`inputFormat`	Format of sequence files. Default input format is "nex", but can also be "fasta" or "seal" (Se-Al). The sequence files must end with one of these suffixes (e.g. *.fas, *.nex, *.seal).

Details

The script "seqConverter.pl" can be found in "/home_path/phrapl/pkg/extdata/"

This perl script was written by Olaf R.P. Bininda-Emonds. For more information see: http://www.uni-oldenburg.de/ibu/systematik-evolutionsbiologie/programme/

Author(s)

Brian O'Meara, Bryan Carstens, Nathan Jackson, Ariadna Morales-Garcia

Maintainer: Brian O'Meara <bomeara@utk.edu>

References

O'Meara, B.C., N. Jackson, A. Morales-Garcia, and B. Carstens (2014) Phrapl in prep.

See Also

RunRaxml

Examples

```
## Not run:
## The script "seqConverter.pl" can be found in "/home_path/phrapl/pkg/extdata/"
inFilePath<-load("/path/sequence_files")

RunSeqConverter(seqConvPath="/phrapl/pkg/extdata/seqConverter.pl",
inFilePath=inFilePath, inputFormat="nex")

## End(Not run)
```

SaveMovie

This function converts a 3D tree plot into an animated gif, suitable for inserting into a slide presentation, putting on a website, etc.

Description

Uses the animation package to save a series of images as a population history rotates, then stitches them to form an animated gif.

Usage

```
SaveMovie(total.revolutions = 1, duration = 10, save.dir = NULL)
```

Arguments

`total.revolutions`	How many full rotations to take.
`duration`	How long should the movie be, in seconds?
`save.dir`	Where should the final movie be saved?

Details

This will require the animation package as well as ImageMagick or GraphicsMagick. It could be modified to create Quicktime or Flash movies instead.

Value

This will create an animated gif in the specified folder.

Author(s)

Brian O'Meara

See Also

`PlotModel`, ~~~

Index

∗ **TestData**
  phrapl-data, 27
  phrapl-example, 28
∗ **~3D**
  PlotModel, 29
  PlotModel2D, 30
∗ **~gridsearch**
  GridSearch, 21
∗ **~grid**
  GridSearch, 21
∗ **~migrationIndividual**
  GenerateMigrationIndividuals, 15
∗ **~movie**
  SaveMovie, 38
∗ **~permutation**
  GetPermutationWeightsAcrossSubsamples, 20
∗ **~plot**
  PlotModel, 29
  PlotModel2D, 30
  SaveMovie, 38
∗ **~subsampling**
  PrepSubsampling, 32
  PrepSubsamplingFiles, 33
∗ **~weights**
  GetPermutationWeightsAcrossSubsamples, 20
∗ **example**
  phrapl-data, 27
  phrapl-example, 28
∗ **package**
  phrapl-package, 2

AddEventToAMigrationArray *(AddEventToMigrationArray)*, 3
AddEventToMigrationArray, 3, *24*
AddGrowthToAMigrationArray, 9
assignFile *(phrapl-example)*, 28

CachedMigrationArrays *(phrapl-data)*, 27
CalculateGDI *(CalculateGdi)*, 9
CalculateGdi, 9
CalculateModelAverages, 10, *12*
ConcatenateResults, *11*, 11
ConvertStructureDataToTrees, 13

example *(phrapl-example)*, 28

GenerateMigrationArrayMap, 14
GenerateMigrationIndividuals, 15
GenerateMigrationIndividualsOneAtATime, 17
GetPermutationWeightsAcrossSubsamples, 20
GridSearch, *7*, *12*, 21

MergeTrees, 25
migrationArray, *14*, *26*
migrationArray *(phrapl-example)*, 28
MigrationArray_3pops_maxMigrationK1 *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK1_asymmetrical *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK2 *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK2_asymmetrical *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK3 *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK3_asymmetrical *(phrapl-data)*, 27
MigrationArray_3pops_maxMigrationK4_asymmetrical *(phrapl-data)*, 27
MigrationArray_4pops_maxMigrationK1 *(phrapl-data)*, 27
MigrationArray_4pops_maxMigrationK2 *(phrapl-data)*, 27
MsIndividualParameters, 26

phrapl *(phrapl-package)*, 2

phrapl-data, 27
phrapl-example, 28
phrapl-package, 2
PlotModel, 29, *38*
PlotModel2D, 30
popAssignments *(*phrapl-example*)*, 28
prep.subsampling *(*PrepSubsampling*)*, 32
prep.subsampling.files
    *(*PrepSubsamplingFiles*)*, 33
PrepSubsampling, 32
Prepsubsampling *(*PrepSubsampling*)*, 32
prepsubsampling *(*PrepSubsampling*)*, 32
PrepSubsamplingFiles, 33
Prepsubsamplingfiles
    *(*PrepSubsamplingFiles*)*, 33
prepsubsamplingfiles
    *(*PrepSubsamplingFiles*)*, 33

ReadStructureData, 34
RunRaxml, 35, *37*
RunSeqConverter, *36*, 37

SaveMovie, *30*, *31*, 38

TestData *(*phrapl-example*)*, 28
trees *(*phrapl-example*)*, 28