

Package ‘cot’

October 1, 2012

Type Package

Title Counting Over Trees

Version 1.0.0

Date 2012-02-01

Author Lukasz Banasiak

Maintainer Lukasz Banasiak <banasiak.luk@gmail.com>

Description This package was developed particularly to count number of dispersals over set of ultrametric (time oriented) phylogenies and do some statistics and plots. But it can be successfully used to count number of changes of any trait on a given ultrametric phylogeny. One limitation is that reconstructions - either biogeographical or morphological - are unequivocal (i.e. every node in a tree is represented by single area). This package can handle really big data sets but doesn't use advantages of multi core/processor machines explicitly. Although some parallelizations can be done manually. Further development will focus on support for not ultrametric trees and equivocal reconstructions. The MapReduce implementation is also available within the open-source project CotMR (<http://>). It covers very similar functionality to 'cot' but is scalable and can be run on Hadoop clusters. 'cot' package can be used to make nice looking plots from CotMR numerical outputs.

License GPL-2

Depends R (>= 2.10), ape, abind, iterators, foreach, TeachingDemos, gtools

LazyLoad yes

R topics documented:

cot-package	2
apioideae	3
array.sort	4
changes.plot	5
changes.plot.csv	6
count.plot	7

count.plot.csv	8
cumul.plot	9
cumul.plot.csv	10
histograms	11
histograms.csv	12
icount.tree	13
load.all	16
migration.hpd	17
parcel.array	18
plot.counts	19
print.counts	19
routes	20
summary.counts	21
test.files.generator	21
trees	22
Index	23

cot-package	<i>Counting Over Trees</i>
-------------	----------------------------

Description

This package was developed particularly to count number of dispersals over set of ultrametric (time oriented) phylogenies and do some statistics and plots. But it can be successfully used to count number of changes of any trait on a given ultrametric phylogeny. One limitation is that reconstructions - either biogeographical or morphological - are unequivocal (i.e. every node in a tree is represented by single area). Of course equivocal reconstructions can be recoded to fit those requirements, i.e. in binary data equivocal reconstruction "0/1" can be coded as third character "2". This package can handle really big data sets but doesn't use advantages of multi core/processor machines explicitly. Although some parallelizations can be done manually. Further development will focus on support for not ultrametric trees and equivocal reconstructions and automatic parallelization as well. The MapReduce implementation is also available within the open-source project CotMR (<http://>). It covers very similar functionality to 'cot' but is scalable and can be run on Hadoop clusters. 'cot' package can be used to make nice looking plots from CotMR numerical outputs.

Details

Package:	cot
Type:	Package
Version:	1.0
Date:	2012-01-11
License:	GPL-2
LazyLoad:	yes

The heart of this package is `icount.tree` function, which counts number of dispersals over set of ultrametric (time oriented) phylogenies and saves results into three dimensional array of class "counts". In such an array indexes of dimensions point to 1) a particular route of dispersal, 2) time and to 3) number of phylogenetic tree in the data set. An array of class "counts" is starting point (input) for many other functions summarizing it in forms of various plots and statistics.

Author(s)

Author and maintainer: Lukasz Banasiak <banasiak.luk@gmail.com>

References

The publication is in the preparation.

apioideae

The Main Tutorial Data Set

Description

This array of class "counts" was built using function `icount.tree` on a sample of trees from Bayesian biogeographical analysis of Apiaceae subfamily Apioideae. The time scale is expressed in million years ago. This data set can be used to see how other functions summarizing effects of counting produced by `icount.tree` work. Those are especially `count.plot`, `cumul.plot`, `changes.plot` and histograms, which produce various plots and statistics based on distribution of dispersals in time and space.

Usage

```
data(apioideae)
```

Format

This is array of class "counts".

Details

This data set covers 10800 trees 1194 taxa each. There should be at least 100 or even better 200 statistically independent trees to make statistical inference unbiased. To see if data set fits this requirements well one can use "Tracer" that calculates Effective Sample Size of set of trees coming from Bayesian analysis prior to running `icount.tree` and then summarizing functions.

Source

The publication relayed on this dataset is in the preparation.

References

Rambaut A, Drummond AJ (2007) Tracer v1.4, Available from <http://beast.bio.ed.ac.uk/Tracer>

Examples

```
data(apioideae)
summary(apioideae)
```

`array.sort`*Merges Arrays of Class "counts" in Alphabetical Order*

Description

This function is highly connected to `load.all`, which loads all the objects from a folder and returns a list of them. If those objects are arrays of class "counts" `array.sort` automatically sorts them alphabetically and combines in one big array of the same class.

Usage

```
array.sort(x)
```

Arguments

`x` A list of arrays of class "counts", especially produced by `load.all`.

Details

This function uses function `mixedsort` from package `gtools` instead of standard `sort` to order the arrays.

Value

Single array of class "counts".

Author(s)

Lukasz Banasiak

See Also

[load.all](#)

changes.plot

*Plots Number of Dispersals Through Time***Description**

This function gets data from an array of class "counts" and generates one of the three types of plots. If the array contains data from more than one tree median and 95 percent HPD intervals are considered. This plot is made as summary for all the routes even if some were not given as an input for `icount.tree` but existed in parsed trees. Function `count.plot` does exactly the same as `changes.plot` with argument `mode = 1` but for each route independently.

Usage

```
changes.plot(a, depth = 0, mode = 0, file = "", ...)
```

Arguments

<code>a</code>	An array of class "counts".
<code>depth</code>	Optional. A numeric value limiting the depth of the plot in time.
<code>mode</code>	An integer selecting type of the plot. 1 is for relative number of dispersals (realised to potential); 2 stands for nominal number of realised dispersals and 3 is for nominal number of potential dispersals.
<code>file</code>	Optional. A character vector defining file name in which plot coordinates will be saved instead of printing on a screen or saving them as "data.frame".
<code>...</code>	Further arguments, passed to plot.

Details

"Realised" dispersals are those that were reconstructed on a tree, i.e consider two nodes connected by a branch and having different areas assigned. In such a case it's assumed that dispersal/change occurred in the deeper node - it is in the very beginning of the branch, which is registered time of dispersal. It means we assume that dispersal is connected to speciation event. Thus the number of "potential" dispersals in particular time slice is just the sum of nodes included in this interval each multiplied by the number of branches that arise from it. It means "potential" dispersals cover those "realised".

Value

Plots the results and prints or saves to a file coordinates of plotted points.

Author(s)

Lukasz Banasiak

See Also

[changes.plot.csv](#) [count.plot](#)

Examples

```
data(apiodeae)
changes.plot(apiodeae, mode=1, depth=18, file="realised2potential.csv")
```

changes.plot.csv	<i>Plots Number of Dispersals Through Time from CSV File</i>
------------------	--

Description

This function gets data from CSV file resulted from either `changes.plot` or MapReduce implementation of this package and makes the plot. If the CSV file contains data from more than one tree median and 95 percent HPD intervals are considered.

Usage

```
changes.plot.csv(file = "", depth = 0, title = "", ylabel = "", ...)
```

Arguments

<code>file</code>	A CSV file resulted from either <code>changes.plot</code> or MapReduce implementation of this package.
<code>depth</code>	Optional. A numeric value limiting the depth of the plot in time.
<code>title</code>	Optional. The main title (character vector) of the plot according to dataset type (relative or nominal). See argument <code>mode</code> in <code>changes.plot</code> for more details.
<code>ylabel</code>	Optional. The y axis label name of the plot according to dataset type (relative or nominal)
<code>...</code>	Further arguments, passed to <code>plot</code> .

Value

Just plots the data.

Author(s)

Lukasz Banasiak

See Also

[changes.plot](#)

Examples

```
data(apioideae)

# Generates a CSV file with data in your working directory.
changes.plot(apioideae, mode=1, depth=18, file="realised2potential.csv")

# Reads the file and plots the data.
changes.plot.csv("realised2potential.csv", depth=18,
  title="Number of realised to potential dispersals through time",
  ylabel="Fraction")
```

count.plot

*Plots Relative Number of Dispersals Through Time***Description**

This function generates relative number of dispersal through time plot for each particular route from an array of class "counts". If the array contains data from more than one tree median and 95 percent HPD intervals are considered.

Usage

```
count.plot(a, depth = 0, title=TRUE, file = "", ...)
```

Arguments

a	An array of class "counts".
depth	Optional. A numeric value limiting the depth of the plot in time.
title	When set to TRUE then automatic main title is generated.
file	Optional. A character vector defining file name in which plot coordinates will be saved instead of printing on a screen or saving in a data.frame.
...	Further arguments, passed to plot.

Details

"Realised" dispersals are those that were reconstructed on a tree, i.e consider two nodes connected by a branch and having different areas assigned. In such a case it's assumed that dispersal/change occurred in the deeper node - it is in the very beginning of the branch, which is registered time of dispersal. Thus the number of "potential" dispersals in particular time slice is just the sum of nodes included in this interval each multiplied by the number of branches that arise from given node. It means "potential" dispersals cover those "realised".

Value

Plots the results and prints or saves to file coordinates of plotted points.

Author(s)

Lukasz Banasiak

See Also[count.plot.csv](#)**Examples**

```
data(apioideae)
par(mfrow=c(3,4))
count.plot(apioideae, depth=18)
```

count.plot.csv

*Plots Relative Number of Dispersals Through Time from CSV File***Description**

This function generates the relative number of dispersal through time plot for each particular route from CSV file resulted from either count.plot or MapReduce implementation of this package. If the CSV file contains data from more than one tree median and 95 percent HPD intervals are considered.

Usage

```
count.plot.csv(file = "", title=TRUE, depth = 0, ...)
```

Arguments

file	A CSV file resulted from either count.plot or MapReduce implementation of this package.
title	When set to TRUE then automatic main title is generated.
depth	Optional. A numeric value limiting the depth of the plot in time.
...	Further arguments, passed to plot.

Value

Just plots the data.

Author(s)

Lukasz Banasiak

See Also[count.plot](#)

Examples

```
data(apioideae)

# Generates a CSV file with data in your working directory.
par(mfrow=c(3,4))
count.plot(apioideae, depth=18, file="migrations.csv")

# Reads the file and plots the data.
par(mfrow=c(3,4))
count.plot.csv("migrations.csv")
```

cumul.plot	<i>Generates the LTT plot</i>
------------	-------------------------------

Description

This function generates Lineages Through Time plot from an array of class "counts". If the array contains data from more than one tree the median and 95 percent HPD are considered.

Usage

```
cumul.plot(a, depth = 0, file = "", ...)
```

Arguments

a	An array of class "counts".
depth	Optional. A numeric value limiting the depth of the plot in time.
file	Optional. A character vector defining file name in which plot coordinates will be saved instead of printing on screen or saving to a file.
...	Further arguments, passed to plot.

Value

Plots the results and prints or saves to file coordinates of plotted points.

Author(s)

Lukasz Banasiak

See Also

[cumul.plot.csv](#)

Examples

```
data(apioideae)
cumul.plot(apioideae, depth=45)
```

cumul.plot.csv	<i>Generates the LTT Plot from CSV File</i>
----------------	---

Description

This function generates Lineages Through Time plot from CSV file resulted from either function `cumul.plot` or MapReduce implementation of this package. If the CSV file contains data from more than one tree median and 95 percent HPD intervals are considered.

Usage

```
cumul.plot.csv(file = "", depth = 0, ...)
```

Arguments

<code>file</code>	A CSV file resulted from either <code>cumul.plot</code> or MapReduce implementation of this package.
<code>depth</code>	Optional. A numeric value limiting the depth of the plot in time.
<code>...</code>	Further arguments, passed to <code>plot</code> .

Value

Just plots the data.

Author(s)

Lukasz Banasiak

See Also

[cumul.plot](#)

Examples

```
data(apioideae)

# Generates a CSV file with data in your working directory.
cumul.plot(apioideae, depth=45, file="ltt.csv")

# Reads the file and plots the data.
cumul.plot.csv("ltt.csv")
```

Description

Generates histograms of Dispersal Asymmetry Index over set of trees for each route. It gets data from an array of class "counts". Additionally it calculates mean DAI and performs test of significance of asymmetry for each route.

Usage

```
histograms(a, precision = 0.1, digits=4, rug=TRUE, title=TRUE, file = "", ...)
```

Arguments

a	An array of class "counts".
precision	A numeric value, which defines how width should be bars of histogram (DAI ranges always from -1 to 1). The default value 0.1 breaks histogram to 20 bars.
digits	An integer giving the number of digits to be printend in output.
rug	If set to TRUE (default) a rug is marked on abscissa axis indicating mean DAI.
title	When set to TRUE then automatic main title is generated.
file	Optional. A character vector defining file stem name in which plot coordinates will be saved instead of printing on a screen or saving to data.frame. Two files will be generated with prefixes "stat." and "hist."
...	Further arguments, passed to hist.

Details

Dispersal Asymmetry Index is a measure of asymmetry that can be calculated for each route and tree. For a particular tree and route it varies from -1 to 1 where value 1 means all migrations were in the dominant direction and -1 that all were in the opposite. The dominant direction of dispersals is calculated according to the whole set of trees. Thus mean value of DAI over set of trees ranges between 0 and 1 where the latter means total asymmetry, it is all dispersals in all trees were in one and only one direction. If 95 percent or more of individual DAIs for particular route were larger than zero one can consider such a route as significantly asymmetric. It means p-value of the test of asymmetry equal or smaller than 0.05.

Value

Plots histograms of DAI distribution over set of trees for each route and saves the coordinates of plot in a file or prints it on a screen. It also rapports about mean DAI for each route and p-value of the test of asymmetry, which are printed on a screen or saved to second file.

Author(s)

Lukasz Banasiak

See Also[histograms.csv](#)**Examples**

```
data(apioideae)
par(mfrow=c(3,4))
histograms(apioideae)
```

histograms.csv*Histograms of DAI from CSV File*

Description

Generates histograms of Dispersal Asymmetry Index over of set of trees. It gets data from CSV file resulted from either histograms or MapReduce implementation of this package.

Usage

```
histograms.csv(file = "", title=TRUE, ...)
```

Arguments

file	A CSV file resulted from either histograms or Apache Hadoop implementation of this package.
title	When set to TRUE then automatic main title is generated.
...	Further arguments, passed to hist.

Value

Just plots the data.

Author(s)

Lukasz Banasiak

See Also[histograms](#)

Examples

```
data(apiodeae)

# Generates a CSV file with data in your working directory.
par(mfrow=c(3,4))
histograms(apiodeae,file="csv")

# Reads the file and generates histograms.
par(mfrow=c(3,4))
histograms.csv("hist.csv")
```

icount.tree

Counts Dispersals over the Ultrametric Phylogenies

Description

This is the basic function of the cot package. It takes for input one or more ultrametric phylogenies in the strict Newick format (optionally object of class "phylo" or "multiPhylo") and a table (a CSV file or object of class "data.frame") of selected pairs of areas (routes). Then it cuts each tree into given time slices and counts number of dispersals in both directions for all selected routes. It returns an array containing numbers of dispersals for each route and tree stratified by time.

Usage

```
icount.tree(tree, routes.list, all.routes = FALSE, states, precision = 5,
division = 0, prefix = "Data", suffix = "A", raport = 10, check = TRUE)
```

Arguments

tree	Either a name of file (character vector) containing a tree or trees in the strict Newick format or object of class "phylo"/"multiPhylo". Only one tree per line is allowed for text file. All nodes should be labeled with particular area including tip labels what means taxon names should be substituted for names of areas. See trees dataset for example.
routes.list	Either a name of CSV file (character vector) containing pairs of selected areas delimited by semicolon between which one wants to count the number of dispersals or its equivalent in "data.frame" class object. Those names have to be exactly the same as used in the tree. See routes dataset for example.
all.routes	A logical value. If set to TRUE routes are not read from CSV file of a data.frame but instead of it all combinations of areas form character vector states are considered.
states	A character vector giving set of areas to be considered. Ignored if all.routes is set to FALSE (default).
precision	A numeric value defining the width of time intervals in which dispersals are counted.

division	An integer used to switch output to set of files rather than single R object stored internally (in both cases array or arrays of class "counts"). The value defines the number of trees from which output is saved to a single file. This option is useful for large tree sets (over 1000 taxa and 5000 trees) to reduce memory usage, speed up computations and save partial results on a disk.
prefix	When division is set greater than zero this stands for prefix for automatically generated output files.
suffix	When division is set greater than zero this stands for suffix for automatically generated output files. Useful when doing parallel analyses in the same directory.
raport	Defines frequency of progress rapport printed to a screen in terms of read trees.
check	When set to TRUE computations are slower but many characteristics of input trees are checked (i.e. if a particular tree is effectively ultrametric) and if something goes wrong such a bad formatted tree is omitted.

Details

The dispersals are assumed to occur on the very beginning of a branch connecting two nodes, which have different areas assigned in the reconstruction. This is how time of dispersal is obtained - it's just time of the deeper node of the branch. It means we assume that dispersal is connected with speciation event.

The specific form of strict Newick format needed for *icount.tree* although canonical is unusual and should be prepared manually. Unfortunately there is no single standard of annotating trees with biogeographical/morphological reconstructions. I recommend writing specific AWK scripts to parse output trees from BEAST - or any other software generating ultrametric phylogenies with reconstructions - to remove comments and substitute taxon names with area labels. Below is simple example of well formatted tree:

```
((A:0.5,B:0.5,C:0.5)C:0.5,(C:0.1,B:0.1)B:0.9,B:1.0)A;
```

More complicated example can be found in trees dataset shipped with this package.

An AWK script for parsing BEAST output can be obtained from author. Further development will focus on implementation this functionality inside *cot* package.

The major limitation is ONLY ONE AREA allowed for each node, i.e. reconstruction cannot be equivocal. It also means that set of areas cannot be assigned to any node what usually means broad distribution of taxon or its ancestor.

Function *icount.tree* accepts polytomies and isn't sensitive for lack of semicolon at the end of line but each line in text file should contain only one tree. If many trees are encountered such a line is omitted from the analysis. One can first read in trees from text file using *read.tree* function from *ape* package. This allows modifications of trees and viewing them but is inefficient with bigger datasets.

This function although doesn't support parallel computing explicit but calculations can be parallelized manually. Results are simply additive and one can divide tree into few minor files/objects and then run analyses separately on different cores/machines and then combine results with *load.all* and *array.sort*.

This package was developed to resolve some biogeographical issues but it can be of course used to count changes of any traits on ultrametric phylogenies. One limitation is that only single state

of a trait is allowed for each node. Any unequivocal parsimony/Maximum Likelihood mapping of categorical data fits those requirements but stochastic mapping and many models of trait evolution implemented in Bayesian framework do as well. At the end one may want for binary data substitute equivocal states "0/1" for third character "2" what solves the problem.

Value

A three dimensional array of class "counts", which contains number of dispersals ordered according to route, time and input tree number.

Author(s)

Lukasz Banasiak <banasiak.luk@gmail.com>

References

Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger, The AWK Programming Language, Addison-Wesley, 1988

Examples

```
# Loads two data sets. First is a data frame with pairs of routes,
# the second is object of class "multiPhylo", which contains trees.
data(routes)
data(trees)

# This performs counting over trees.
Data <- icount.tree(trees, routes, raport=1)
summary(Data)

# Generates two text files in your working directory:
# "Test.10.txt" and "Routes.csv". First contains sample of trees
# and the latter table of routes. Watch them carefully to get familiar
# with input format.
data(routes)
write.table(routes, file="Routes.csv", col.names=FALSE,
            row.names=FALSE, sep=";", quote=FALSE)
rm(routes)

data(trees)
write.tree(trees, file="Test.25.txt")
rm(trees)

# This performs counting over trees.
Data <- icount.tree("Test.25.txt", "Routes.csv", raport=1)
summary(Data)

# Generates 10 random trees in object
# of class "multiPhylo" with areas designated from A to E
# using functionalities provided by "ape" package.
trees <- foreach(i=1:10) %do%
```

```

{
  tree <- rbdtree(0.05,0)
  tree$tip.label <- sample(LETTERS[1:5],length(tree$tip.label),replace=TRUE)
  tree$node.label <- sample(LETTERS[1:5],tree$Nnode,replace=TRUE)
  tree
}
class(trees) <- "multiPhylo"

# Plots first tree with axis
plot(trees[[1]], show.node.label=TRUE)
axisPhylo()

# Does counting over those 10 trees
Data <- icount.tree(trees, states=LETTERS[1:5], all.routes=TRUE)
summary(Data)

```

load.all

*Loads All the Objects from a Folder***Description**

This function loads all the objects from a working directory and returns a list of those objects. It is particularly useful when `icount.tree` was run with division argument set greater than zero and many output files containing arrays of class "counts" were then produced. It is strongly connected to `array.sort`, which sorts elements of such a list and combines them into one big array of class "counts".

Usage

```
load.all()
```

Details

Function has no arguments and tries to load all the files from a directory irrespectively to classes of objects enclosed in .Rda archives. It can be used as general purpose function.

Value

A list of all loaded objects.

Author(s)

Lukasz Banasiak

See Also

[array.sort](#) [icount.tree](#)

migration.hpd*Calculates 95 Percent HPD and Median for Routes*

Description

This function calculates 95 percent Highest Posterior Density and median for each one-way route included in an input array of class "counts" over the whole time scale (merging time intervals into single one).

Usage

```
migration.hpd(a, in.pairs=TRUE, hist=FALSE, file = "", ...)
```

Arguments

<code>a</code>	An array of class "counts".
<code>in.pairs</code>	A logical value. If set TRUE output is more detailed - each track is divided into two one-way routes.
<code>hist</code>	A logical value indicating whether histograms of calculated statistics should be plotted or not.
<code>file</code>	Optional. A character vector containing file name in which results are saved instead of building a data.frame.
<code>...</code>	Further arguments, passed to hist.

Value

A data.frame, which can be saved automatically to CSV file when "file" argument is specified.

Author(s)

Lukasz Banasiak

Examples

```
data(apiodeae)
migration.hpd(apiodeae)
```

`parcel.array`*Divides an Array of Class "counts" to Smaller Parts*

Description

This function was developed to decrease memory usage on older systems. Arrays of class "counts" can be really large objects if number of routes and/or number of analyzed trees with `icount.tree` was prominent. Function `parcel.array` divides such an array to smaller parts but in each keeps "header" and thus preserves class structure. Parts contain limited number of routes considered but each has the full time scale.

Usage

```
parcel.array(a, chunks = 1, prefix = "Data", suffix = "A")
```

Arguments

<code>a</code>	An array of class "counts".
<code>chunks</code>	An integer defining how many routes should contain each part.
<code>prefix</code>	A prefix to object's and file's name of particular parts.
<code>suffix</code>	A suffix to object's and file's name of particular parts.

Details

Note that obtained parts by `parcel.array` cannot be merged by `array.sort`, which is used to combine arrays with the same set of routes.

Value

It saves parts of divided array to files in working directory.

Author(s)

Lukasz Banasiak

Examples

```
data(apiodeae)
parcel.array(apiodeae, chunks=5, prefix = "Part", suffix = "")
```

`plot.counts`*Guides to Plotting Functions*

Description

A generic function used usually to plot results but in this case just prints names of specific functions that provide various types of plots for an array of class "counts".

Usage

```
## S3 method for class 'counts'  
plot(x, ...)
```

Arguments

<code>x</code>	An array of class "counts"
<code>...</code>	Further arguments to be passed to print.

Value

Just prints the tips.

Author(s)

Lukasz Banasiak

Examples

```
data(apiidaeae)  
plot(apiidaeae)
```

`print.counts`*Prints an Array of Class "counts"*

Description

A generic function, which prints array of class "counts".

Usage

```
## S3 method for class 'counts'  
print(x, range = 1, ...)
```

Arguments

<code>x</code>	An array of class "counts".
<code>range</code>	An integer vector pointing from which trees information should be printed. Default is to print information only from first tree.
<code>...</code>	Further arguments passed to print.

Value

Prints result on a screen.

Author(s)

Lukasz Banasiak

Examples

```
data(apioideae)
print(apioideae)
```

routes

The Tutorial Data Set with Pairs of Areas (Routes)

Description

The tutorial data set with pairs of areas (routes) compatible with dataset trees. Both should be used to run an example of `icount.tree` function.

Usage

```
data(apioideae)
```

Format

This is CSV file saved as data.frame. See examples how to restore original file.

Source

The publication relayed on this dataset is in the preparation.

Examples

```
# Restores original CSV file in working directory
data(routes)
write.table(routes, file="Routes.csv", col.names=FALSE,
            row.names=FALSE, sep=";", quote=FALSE)
```

summary.counts	<i>Summarizes an Array of Class "counts"</i>
----------------	--

Description

A generic function, which summarizes an array of class "counts".

Usage

```
## S3 method for class 'counts'
summary(object, ...)
```

Arguments

object	An array of class "counts".
...	Further arguments, passed to print.

Value

Prints result on a screen.

Author(s)

Lukasz Banasiak

Examples

```
data(apiodeae)
summary(apiodeae)
```

test.files.generator	<i>Performs Full Set of Analyses</i>
----------------------	--------------------------------------

Description

This function performs set of analyses as described in original publication for biogeography of Apiaceae subfamily Apioideae.

Usage

```
test.files.generator(a, depth = 0)
```

Arguments

a	An array of class "counts".
depth	Optional. A numeric value limiting the depth of the analyses in time.

Value

Plots are generated and statistics are saved to files in working directory. Names of files are reported on a screen.

Author(s)

Lukasz Banasiak

References

The publication is in the preparation.

Examples

```
data(apiodeae)
test.files.generator(apiodeae, depth=18)
```

trees

The Tutorial Data Set with Sample of Trees

Description

The tutorial data set of 25 phylogenies of 1194 members of Apiaceae subfamily Apioideae compatible with dataset routes. Both should be used to run an example of `icount.tree` function.

Usage

```
data(apiodeae)
```

Format

This is object of class "multiPhylo". See examples how to restore original text file.

Source

The publication relayed on this dataset is in the preparation.

Examples

```
# Restores original text file in working directory
data(trees)
write.tree(trees, file="Test.25.txt")
```

Index

- *Topic **DAI**
 - histograms, [11](#)
 - histograms.csv, [12](#)
- *Topic **chain**
 - test.files.generator, [21](#)
- *Topic **combine**
 - array.sort, [4](#)
- *Topic **count**
 - cot-package, [2](#)
 - icount.tree, [13](#)
- *Topic **datasets**
 - apioideae, [3](#)
 - routes, [20](#)
 - trees, [22](#)
- *Topic **divide**
 - parcel.array, [18](#)
- *Topic **histogram**
 - histograms, [11](#)
 - histograms.csv, [12](#)
- *Topic **load**
 - load.all, [16](#)
- *Topic **ltt**
 - cumul.plot, [9](#)
 - cumul.plot.csv, [10](#)
- *Topic **merge**
 - array.sort, [4](#)
- *Topic **package**
 - cot-package, [2](#)
- *Topic **phylogeny**
 - cot-package, [2](#)
- *Topic **plot**
 - changes.plot, [5](#)
 - changes.plot.csv, [6](#)
 - count.plot, [7](#)
 - count.plot.csv, [8](#)
 - plot.counts, [19](#)
- *Topic **print**
 - print.counts, [19](#)
 - summary.counts, [21](#)
- *Topic **sort**
 - array.sort, [4](#)
- *Topic **summary**
 - migration.hpd, [17](#)
 - print.counts, [19](#)
 - summary.counts, [21](#)
- apioideae, [3](#)
- array.sort, [4](#), [16](#)
- changes.plot, [5](#), [6](#)
- changes.plot.csv, [5](#), [6](#)
- cot (cot-package), [2](#)
- cot-package, [2](#)
- count.plot, [5](#), [7](#), [8](#)
- count.plot.csv, [8](#), [8](#)
- cumul.plot, [9](#), [10](#)
- cumul.plot.csv, [9](#), [10](#)
- histograms, [11](#), [12](#)
- histograms.csv, [12](#), [12](#)
- icount.tree, [13](#), [16](#)
- load.all, [4](#), [16](#)
- migration.hpd, [17](#)
- parcel.array, [18](#)
- plot.counts, [19](#)
- print.counts, [19](#)
- routes, [20](#)
- summary.counts, [21](#)
- test.files.generator, [21](#)
- trees, [22](#)