Package 'ClockstaR2'

March 25, 2014

Type Package

Title ClockstaR: choosing the number of relaxed-clock models in molecular phylogenetic analysis

Version 2.0

Date 2014-03-21

Author Sebastian Duchene

Maintainer Sebastian Duchene <sebastian.duchene@sydney.edu.au>

Description Relaxed molecular clocks allow the phylogenetic estimation of evolutionary timescales even when substitution rates vary among branches. In analyses of large multigene datasets, it is often appropriate to use multiple relaxed-clock models to accommodate differing patterns of rate variation among genes. We present ClockstaR, a method for selecting the number of relaxed clocks for multigene datasets.

Depends R (>= 2.15.0), ape (>= 3.0-8), cluster (>= 1.14.4)

Suggests phangorn (>= 1.7-4), doParallel (>= 1.0.3), foreach(>= 1.4.1)

License GNU

R topics documented:

ClockstaR2-package *ClockstaR: choosing the number of relaxed-clock models in molecular phylogenetic analysis*

Description

Relaxed molecular clocks allow the phylogenetic estimation of evolutionary timescales even when substitution rates vary among branches. In analyses of large multigene datasets, it is often appropriate to use multiple relaxed-clock models to accommodate differing patterns of rate variation among genes. We present ClockstaR, a method for selecting the number of relaxed clocks for multigene datasets.

Details

Package:	ClockstaR2
Type:	Package
Version:	2.0
Date:	2014-03-21
License:	GNU

Author(s)

Sebastian Duchene Maintainer: Sebastian Duchene <sebastian.duchene@sydney.edu.au>

References

Duchêne, S, Molak, M., and Ho, SYW. "ClockstaR: choosing the number of relaxed-clock models in molecular phylogenetic analysis." Bioinformatics (2013): btt665.

See Also

NONE.

Examples

```
# Create a list of trees of class multiPhylo with four patterns of among-lineage rate variation
tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))
```

```
trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}

names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix(trees.list)

# Find the optimal number of partitions:
partitions.object <- partitions(trees.bsd)

# plot partitions in two graphics devices
plot(partitions.object)
```

`bsd.dist` *Estimate sBSDmin distnace between a pair of trees*

Description

This function estimates the sBSDmin distance between a pair of trees. This distance metric is described in the original publication. See reference bellow.

Usage

```
bsd.dist(tree1, tree2)
```

Arguments

`tree1`	A phylogenetic tree of class 'phylo'. It should have branch lengths.
`tree2`	A phylogenetic tree of class 'phylo'. It should have branch lengths.

Details

See reference

Value

A numeric vector with the sBSDmin (min.bdi.scaled), the scaling factor (scaling.factor), and the unscaled BSD (min.bdi).

Author(s)

Sebastian Duchene

References

Duchêne, S., Molak, M., and Ho, SYW. "ClockstaR: choosing the number of relaxed-clock models in molecular phylogenetic analysis." Bioinformatics (2013): btt665.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

# Simulate two trees with identical topologies, but different patterns of among lineage rate variation

tr1 <- rtree(10)

tr2 <- tr1
tr2$edge.length <- sample(tr1$edge.length)

par(mfrow = c(1, 2))
plot(tr1)
plot(tr2)

bsd.dist(tr1, tr2)


## The function is currently defined as
function (tree1, tree2)
{
    list.tr <- list()
    list.tr[[1]] <- tree1
    list.tr[[2]] <- tree2
    lens <- c(sum(tree1$edge.length), sum(tree2$edge.length))
    tree1 <- list.tr[lens == max(lens)][[1]]
    tree2 <- list.tr[lens == min(lens)][[1]]
    tree.dist.opt <- function(x) {
        tree3 <- tree2
        tree3$edge.length <- tree2$edge.length * x
        return(dist.topo(tree1, tree3, method = "score"))
    }
    opt.dist <- optim(0, fn = tree.dist.opt, method = "Brent",
        lower = 0, upper = 50)
    min.bdi <- opt.dist$value
    scaling <- opt.dist$par
    tree2.scaled <- tree2
    tree2.scaled$edge.length <- tree2$edge.length * scaling
    root.scaling <- 0.05/mean(c(tree1$edge.length[tree1$edge.length >
        1e-05], tree2.scaled$edge.length[tree2.scaled$edge.length >
        1e-05]))
    tree1.root.scaled <- tree1
    tree2.root.scaled <- tree2.scaled
    tree1.root.scaled$edge.length <- tree1$edge.length * root.scaling
    tree2.root.scaled$edge.length <- tree2.scaled$edge.length *
        root.scaling
```

```
  min.bdi.root.scaled <- dist.topo(tree1.root.scaled, tree2.root.scaled,
      method = "score")
  res.vect <- c(min.bdi.root.scaled, scaling, min.bdi)
  names(res.vect) <- c("min.bdi.scaled", "scaling.factor",
      "min.bdi")
  return(res.vect)
}
```

`bsd.matrix` *Estimate the sBSDmin distance for all pairs of trees in a list*

Description

bsd.matrix estimates the sBSDmin distance for all paris of trees in a list. It returns an object of class bsd with the pairswise distances and the scaling factors. The scaling factors represent the differences in the magnitude of the rates among trees. See the help file for bsd.dist for more details.

Usage

```
bsd.matrix(tree.list)
```

Arguments

`tree.list`	A list of trees. It can be a list where each item is a tree, or an object of class multiPhylo. The trees can have names. See the example bellow.

Value

An object of class 'bsd', which is a list with

`comp1`	An object of class dist with the pairwise distances for trees
`comp2`	An object of class matrix with the scaling factors among trees

Author(s)

Sebastian Duchene

See Also

bsd.dist

Examples

```
# Create a list of trees of class multiPhylo with four patterns of among-lineage rate variation
tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
```

```
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))

trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}

names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix(trees.list)


## The function is currently defined as
function (tree.list)
{
    d.mat <- matrix(NA, nrow = length(tree.list), ncol = length(tree.list))
    rownames(d.mat) <- names(tree.list)
    colnames(d.mat) <- names(tree.list)
    s.mat <- d.mat
    print("Estimating tree distances")
    if (length(tree.list) > 3) {
        d.mat.lin <- vector()
        d.mat.lin <- sapply(2:nrow(d.mat), function(a) {
            print(paste("estimating distances", a - 1, "of",
                nrow(d.mat) - 1))
            lapply(tree.list[1:(a - 1)], function(y) {
                bsd.dist(tree1 = y, tree2 = tree.list[[a]])
            })
        })
        for (a in 1:length(d.mat.lin)) {
            vec.temp.dist <- vector()
            vec.temp.scale <- vector()
            for (b in 1:length(d.mat.lin[[a]])) {
                vec.temp.dist[b] <- d.mat.lin[[a]][[b]][1]
                vec.temp.scale[b] <- d.mat.lin[[a]][[b]][2]
            }
            d.mat[a + 1, 1:length(vec.temp.dist)] <- vec.temp.dist
            s.mat[a + 1, 1:length(vec.temp.dist)] <- vec.temp.scale
        }
    }
    else if (length(tree.list) <= 3) {
        stop("The number of gene trees is <= 3. ClockstaR requires at least gene 4 trees")
    }
```

```
  d.mat.lin <- bsd.dist(tree.list[[1]], tree.list[[2]])
  d.mat[2, 1] <- d.mat.lin[1]
  s.mat[2, 1] <- d.mat.lin[2]
  res.list <- list()
  res.list[[1]] <- as.dist(d.mat)
  res.list[[2]] <- s.mat
  class(res.list) <- "bsd"
  return(res.list)
}
```

`bsd.matrix.para` *Estimate the sBSDmin distance for all pairs of trees in a list with parallelisation.*

Description

bsd.matrix estimates the sBSDmin distance for all paris of trees in a list. It returns an object of class bsd with the pairswise distances and the scaling factors. The scaling factors represent the differences in the magnitude of the rates among trees. See the help file for bsd.dist for more details.

Usage

```
bsd.matrix.para(tree.list, para = F, ncore = 1)
```

Arguments

`tree.list`	A list of trees. It can be a list where each item is a tree, or an object of class multiPhylo. The trees can have names. See the example bellow.
`para`	logical. select T to run in parallel.
`ncore`	number of cores for parallelisation.

Value

An object of class 'bsd', which is a list with

`comp1`	An object of class dist with the pairwise distances for trees
`comp2`	An object of class matrix with the scaling factors among trees

Author(s)

Sebastian Duchene

See Also

bsd.dist bsd.matrix

Examples

```
# Create a list of trees of class multiPhylo with four patterns of among-lineage rate variation
tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))

trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}

names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

## Not run:
# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix.para(trees.list, para = T, ncore = 2)

## End(Not run)


## The function is currently defined as
function (tree.list, para = F, ncore = 1)
{
require(foreach)
require(doParallel)

    if (length(tree.list) <= 3) {
        stop("The number of gene trees is <= 3. ClockstaR requires at least gene 4 trees")
    }
    bsd.dist <- function(tree1, tree2) {
        list.tr <- list()
        list.tr[[1]] <- tree1
        list.tr[[2]] <- tree2
        lens <- c(sum(tree1$edge.length), sum(tree2$edge.length))
        tree1 <- list.tr[lens == max(lens)][[1]]
        tree2 <- list.tr[lens == min(lens)][[1]]
        tree.dist.opt <- function(x) {
```

```
        tree3 <- tree2
        tree3$edge.length <- tree2$edge.length * x
        return(dist.topo(tree1, tree3, method = "score"))
    }
    opt.dist <- optim(0, fn = tree.dist.opt, method = "Brent",
        lower = 0, upper = 50)
    min.bdi <- opt.dist$value
    scaling <- opt.dist$par
    tree2.scaled <- tree2
    tree2.scaled$edge.length <- tree2$edge.length * scaling
    root.scaling <- 0.05/mean(c(tree1$edge.length[tree1$edge.length >
        1e-05], tree2.scaled$edge.length[tree2.scaled$edge.length >
        1e-05]))
    tree1.root.scaled <- tree1
    tree2.root.scaled <- tree2.scaled
    tree1.root.scaled$edge.length <- tree1$edge.length *
        root.scaling
    tree2.root.scaled$edge.length <- tree2.scaled$edge.length *
        root.scaling
    min.bdi.root.scaled <- dist.topo(tree1.root.scaled, tree2.root.scaled,
        method = "score")
    res.vect <- c(min.bdi.root.scaled, scaling, min.bdi)
    names(res.vect) <- c("min.bdi.scaled", "scaling.factor",
        "min.bdi")
    return(res.vect)
}
sub.trees <- list()
for (k in 2:length(tree.list)) {
    sub.trees[[k]] <- tree.list[1:k - 1]
}
compute.tree.dists <- function(tree.sub.list, fix.tree) {
    res <- sapply(tree.sub.list, function(a) {
        return(bsd.dist(fix.tree, a))
    })
    return(res)
}
if (para == T) {
    cl <- makeCluster(ncore)
    registerDoParallel(cl)
    print(paste("Clusters registered as follows: ", cl))
    res.par <- foreach(s.trees = sub.trees, j = 1:length(tree.list),
        .packages = "ape") %dopar% compute.tree.dists(tree.sub.list = s.trees,
        fix.tree = tree.list[[j]])
    stopCluster(cl)
}
else if (para == F) {
    res.par <- foreach(s.trees = sub.trees, j = 1:length(tree.list),
        .packages = "ape") %do% compute.tree.dists(tree.sub.list = s.trees,
        tree.list[[j]])
}
res.list <- list()
res.list[[1]] <- matrix(NA, nrow = length(tree.list), ncol = length(tree.list))
for (m in 2:nrow(res.list[[1]])) {
    res.list[[1]][m, 1:ncol(res.par[[m]])] <- res.par[[m]][1,
        ]
}
rownames(res.list[[1]]) <- names(tree.list)
```

```
    colnames(res.list[[1]]) <- names(tree.list)
    res.list[[1]] <- as.dist(res.list[[1]])
    res.list[[2]] <- matrix(NA, nrow = length(tree.list), ncol = length(tree.list))
    for (m in 2:nrow(res.list[[2]])) {
        res.list[[2]][m, 1:ncol(res.par[[m]])] <- res.par[[m]][2,
            ]
    }
    rownames(res.list[[2]]) <- names(tree.list)
    colnames(res.list[[2]]) <- names(tree.list)
    class(res.list) <- "bsd"
    return(res.list)
}
```

`clockstar.interactive` *Run clockstar interactively*

Description

This function can be used to run ClockstaR interactively. It prompts the use for a file with the gene trees in NEWICK format and some options for the clustering methods.

Usage

```
clockstar.interactive()
```

Author(s)

Sebastian Duchene

References

Duchêne, S, Molak, M., and Ho, SYW. "ClockstaR: choosing the number of relaxed-clock models in molecular phylogenetic analysis." Bioinformatics (2013): btt665.

See Also

optim.trees.interactive() to obtain the gene trees.

Examples

```
## Not run:
clockstar.interactive()

## End(Not run)

## The function is currently defined as
function ()
{
    print("Welcome to ClockstaR2")
    trees.file <- readline("please drag or type in the path to your gene trees file in NEWICK format:\n")
    print(paste("reading trees from file: ", trees.file))
    trees <- read.tree(trees.file)
    print(paste("I read", length(trees), "from your file"))
    if (length(names(trees) > 0)) {
```

```
    print(paste("The names of your trees are:"))
    print(names(trees))
}
else {
    warning("These trees have no names")
}
if (("foreach" %in% installed.packages()[, 1]) && "doParallel" %in%
    installed.packages()[, 1]) {
    print("Packages foreach and doParallel are available for parallel computation")
    para <- readline("Should we run ClockstaR in parallel (y / n)? (This is good for large data sets)\n
    if (para == "y") {
        require(foreach)
        require(doParallel)
        ncore <- as.integer(readline("How many cores should ClockstaR use (integer):\n"))
        print(paste("running ClockstaR in parallel with",
            ncore, "cores"))
        trees.bsd <- bsd.matrix.para(trees, para = T, ncore = ncore)
    }
    else {
        print("Calculating sBSDmin distances between all pairs of trees")
        trees.bsd <- bsd.matrix(trees)
    }
}
else {
    print("Calculating sBSDmin distances between all pairs of trees")
    trees.bsd <- bsd.matrix(trees)
}
print("I finished calculating the sBSDmin distances between trees\n")
default.run <- readline("The settings for clustering with ClockstaR are:\nPAM clustering algorithm\nK f
if (default.run == "y") {
    part.data <- partitions(trees.bsd)
}
else {
    fun.cluster <- "d"
    while (!(fun.cluster %in% c("1", "2", "3"))) {
        fun.cluster <- readline("Please select one of the clustering functions bellow:(1-3)\n(1) PAM\n(
    }
    if (fun.cluster == "1") {
        fun <- pam
    }
    else if (fun.cluster == "2") {
        fun <- clara
    }
    else if (fun.cluster == "3") {
        fun <- fanny
    }
    max.k <- as.numeric(readline("What should be the maximum k to test (the maximum is the number of da
    if (max.k >= ncol(trees.bsd[[2]])) {
        stop("The maximun k should be between 1 and the number of data sets - 1. ABORTING")
    }
    criterion <- readline("Please type in the criterion to select the optimal k (This can be firstSEmax
    while (!(criterion %in% c("firstSEmax", "Tibs2001max",
        "globalSEmax", "firstmax", "globalmax"))) {
        criterion <- readline("You have not selected any of the available criteria. Please select again
    }
    boot.n <- as.numeric(readline("How many boostrap replicates should be run for the Gap statistic?\n"
    print("The settings for clustering are complete. The settings are:\n")
```

```
        print("cluster function, maximum k, criterion")
        print(c(c("PAM", "CLARA", "FANNY")[as.numeric(fun.cluster)],
            max.k, criterion))
        part.data <- partitions(trees.bsd, FUN = fun, kmax = max.k,
            B = boot.n, gap.best = criterion)
    }
    print("ClockstaR has finished running")
    print(paste("The best number of partitions for your data set is:",
        part.data$best.k))
    save.dat <- readline("Do you wish to save the results in a pdf file?(y/n)\n")
    if (save.dat == "y") {
        fil.name <- readline("What should be the name and path of the output file?\n")
        plot.partitions(part.data, save.plot = T, file.name = fil.name)
    }
    else {
        print("please see the results in the active graphics devices")
        plot.partitions(part.data)
    }
    return(part.data)
}
```

`optim.trees.interactive` *Optimise the branch lengths of gene trees interactively*

Description

optim.tree.interactive can be used to optimise the gene trees interactively. The data should be a folder with the alignment for each data subset (or gene) and a tree topology. The alignments should be in FASTA format and the tree should be in NEWICK. Please see the example data set.

Usage

```
optim.trees.interactive()
```

Details

The function does not take any arguments. When it is called, it prompts the user for the folder with the data and the substitution models for each data set.

Author(s)

Sebastian Duchene

Examples

```
## Not run:
optim.trees.interactive() # Follow the instructions in the prompt

## End(Not run)
```

partitions *function partitions applied to objects of class bsd*

Description

This is the definition of method partitions for bsd objects. See partitions.bsd for more details.

Usage

```
partitions(bsd.object, ...)
```

Arguments

bsd.object	An object of class bsd, obtained with bsd.matrix or bsd.matrix.para

Author(s)

Sebastian Duchene

See Also

partitions.bsd

Examples

```
## Not run:

tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))

trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}
```

```
names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix(trees.list)

partitions(trees.bsd)



## End(Not run)

## The function is currently defined as
function (bsd.object, ...)
UseMethod("partitions")
```

`partitions.bsd` *Obtain optimal partitioning scheme*

Description

This function obtains the optimal number of partitions for gene trees. The input should be an object of class 'bsd' with the pairwise distances among tres and the scaling factors (see bsd.matrix for more details on this class). partitions.bsd returns an object of class partitions with a matrix with the parition assignments, the range of the sBSDmin distances and an object of class cluster with the output from the clustering analysis.

Usage

```
partitions.bsd(bsd.object, FUN = pam, find.best = T, B = 500, gap.best = "firstSEmax", kmax = ""
```

Arguments

`bsd.object`	An object of class bsd. It can be obtained witn bsd.matrix or bsd.matrix.para. The example(bsd.matrix) for details.
`FUN`	A clustering function as defined in the package cluster. It can be pam, clara, or fanny. However note that for fanny the maximum k should be the number of data subsets / 2.
`find.best`	logical. Select T to find the optimal number of partitions. If F, the function does not find the best number of partitions.
`B`	Numeric. Number of bootstrap replicates for the Gap statistic. Not used if find.best == F
`gap.best`	A character. The criterion to select the optimal number of partitions (k). It can be any of "firstSEmax", "Tibs2001SEmax", "globalSEmax", "firstmax", or "globalmax".
`kmax`	Numeric. Maximum number of partitions to test. The default is the number of data subsets - 1.
`...`	Additional arguments passed to the clustering function.

Details

see the help for pacakge clustering for more details.

Value

An object of class 'partitions'

parts.mat	A matrix with the cluster assignments. The partitions are numbered from 1:maximum k. The columns are each value of k, and the columns are the names of the data subsets if provided.
range.bsd	A numeric vector with the range of the sBSDmin distances
best.k	The optimal number of partitions. Only returned if find.best == T
clusterdata	Results from the Gap statistic. Only returned if find.best == T. See ?clusGap for more details

Author(s)

Sebastian Ducnene

References

Tibshirani, R., Walther, G. and Hastie, T. (2001). Estimating the number of data clusters via the Gap statistic. _Journal of the Royal Statistical Society B_, *63*, 411-423.

Tibshirani, R., Walther, G. and Hastie, T. (2000). Estimating the number of clusters in a dataset via the Gap statistic. Technical Report. Stanford.

Per Broberg (2006). SAGx: Statistical Analysis of the GeneChip. R package version 1.9.7. <URL: http://home.swipnet.se/pibroberg/expression_hemsida1.html>

Examples

```
# Create a list of trees of class multiPhylo with four patterns of among-lineage rate variation
tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))

trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
```

```
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}

names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix(trees.list)

# Find the optimal number of partitions:
partitions.object <- partitions(trees.bsd)

# plot partitions in two graphics devices
plot(partitions.object)




## The function is currently defined as
function (bsd.object, FUN = pam, find.best = T, B = 500, gap.best = "firstSEmax",
    kmax = "", ...)
{
    dimat <- as.matrix(bsd.object[[1]])
    if (kmax == "") {
        kmax <- nrow(dimat) - 1
    }
    if (find.best == T) {
        clusdat <- clusGap(dimat, B = B, FUNcluster = FUN, K.max = kmax)
        npart <- maxSE(f = clusdat$Tab[, 3], SE.f = clusdat$Tab[,
            4], method = gap.best)
    }
    parts.list <- list()
    for (i in 1:kmax) {
        clus.temp <- FUN(dimat, k = i, ...)
        parts.list[[i]] <- clus.temp$clustering
    }
    parts.mat <- do.call("cbind", parts.list)
    rownames(parts.mat) <- rownames(dimat)
    colnames(parts.mat) <- paste0("k=", 1:ncol(parts.mat))
    res <- list(parts.mat, range(bsd.object[[1]]))
    names(res) <- c("parts.mat", "range.bsd")
    if (exists("npart")) {
        colnames(res[[1]])[npart] <- paste0(colnames(parts.mat)[npart],
            "BEST")
        res[[3]] <- npart
        names(res)[3] <- "best.k"
    }
    if (find.best == T) {
        res[[4]] <- clusdat
        names(res)[4] <- "clusterdata"
    }
    class(res) <- "partitions"
    return(res)
  }
```

plot.partitions *plot objects of class partitions*

Description

This function plots the results from the gap statistic from package cluster, and a matrix with colours for the partition assignments.

Usage

```
plot.partitions(partitions.object, save.plot = F, file.name = "results_clockstar")
```

Arguments

partitions.object	An object of class 'partitions', obtained with partitions.bsd
save.plot	logical. Select T to save the plots to a pdf file, or F to print in two new graphics devices.
file.name	A character. If save.plot == T, the name of the file to print the results.

Author(s)

Sebastian Duchene

Examples

```
# Create a list of trees of class multiPhylo with four patterns of among-lineage rate variation
tr.fix <- rtree(10)

set.seed(12345)
rates1 <- abs(rnorm(18, sd = 0.1))
set.seed(123456)
rates2 <- abs(rnorm(18, sd = 0.1))
set.seed(1234567)
rates3 <- abs(rnorm(18, sd = 0.1))
set.seed(12345678)
rates4 <- abs(rnorm(18, sd = 0.1))

trees.list <- list()

for(i in 1:20){
      trees.list[[i]] <- tr.fix
      if(i <= 5){
          trees.list[[i]]$edge.length <- abs(rates1 + rnorm(18, 0, 0.01))
      }else if(i > 5 && i <= 10){
           trees.list[[i]]$edge.length <- abs(rates2 + rnorm(18, 0, 0.01))
      }else if(i >= 10 && i < 15){
           trees.list[[i]]$edge.length <- abs(rates3 + rnorm(18, 0, 0.01))
      }else{
    trees.list[[i]]$edge.length <- abs(rates4 + rnorm(18, 0, 0.01))
      }
}
```

```
names(trees.list) <- paste0("tree", 1:20)
class(trees.list) <- "multiPhylo"

# Estimate sBSDmin distance for all pairs of trees:
trees.bsd <- bsd.matrix(trees.list)

# Find the optimal number of partitions:
partitions.object <- partitions(trees.bsd)

# plot partitions in two graphics devices
plot(partitions.object)
```

```
## The function is currently defined as
function (partitions.object, save.plot = F, file.name = "results_clockstar")
{
    image(t(partitions.object$parts.mat), col = sample(rainbow(ncol(partitions.object$parts.mat))),
        axes = F, ylab = "Data subset", main = "Partitioning scheeme for values of k")
    mtext(text = rownames(partitions.object$parts.mat), side = 2,
        line = 0.3, at = seq(0, 1, 1/(nrow(partitions.object$parts.mat) -
            1)), las = 1, cex = 0.6)
    mtext(text = colnames(partitions.object$parts.mat), side = 3,
        line = 0.3, at = seq(0, 1, 1/(ncol(partitions.object$parts.mat) -
            1)), las = 1, cex = 0.6)
    if (save.plot) {
        dev.copy2pdf(file = paste0(file.name, "_matrix.pdf"))
    }
    dev.new()
    plot(partitions.object$clusterdata)
    if (save.plot) {
        dev.copy2pdf(file = paste0(file.name, "_gapstats.pdf"))
        sapply(dev.list(), function(x) dev.off(x))
    }
  }
```

Index