Package 'clockstar'

January 26, 2014

Type Package

Title ClockstaR: A package for molecular clock partition selection

Version 0.1

Date 2013-12-07

Author Sebastian Duchene, Martyna Molak, and Simon Y. W. Ho.

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

Description This package implements a standardised branch distance score for phylogenetic trees to assess the best molecular clock partitioning strategy, given the patterns of among-lineage rate variation in multi-gene data sets.

Suggests ape, phangorn, geiger, tcltk, foreach, doParallel, rgl,igraph, Matrix, cluster, Hmisc

Depends ape, phangorn, geiger, tcltk, foreach, doParallel, rgl,igraph, Matrix, cluster, Hmisc

Imports ape, phangorn, geiger, tcltk, foreach, doParallel, rgl,igraph, Matrix, cluster, Hmisc

License GPL

R topics documented:

clockstar-package *ClockstaR: Selecting partitioning strategies for relaxed phylogenetic analysis*

Description

This package implements the method described in Duchene et al. It assigns multigene data subsets to molecular clock partitions in relaxed molecular clock analysis.

Details

Package:	clockstar
Type:	Package
Version:	1.0
Date:	2013-10-02
License:	What license is it under?

Please see the user manual for a tutorial on how to run ClockstaR

Author(s)

Sebastian Duchene

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

References

Duchene et al. Duchene et al.

See Also

Please visit the website bellow for more tutorials and more information http://sydney.edu.au/science/biology/meep/softwa

Examples

```
# This is an example of using ClockstaR in automatic mode.

# generate a list of trees with three patterns of among-lineage rate variation

fixed.topology <- rtree(10)
fixed.topology$edge.length <- NULL
tree.list <- list()
set.seed(1234)
patterns1 <- abs(rnorm(18, 0, 10))
set.seed(3456)
patterns2 <- abs(rnorm(18, 0, 10))
set.seed(0987)
patterns3 <- abs(rnorm(18, 0, 10))
for(i in 1:3){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns1 + rnorm(18))
}
```

```
for(i in 4:7){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns2 + rnorm(18))
}

for(i in 8:10){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns3 + rnorm(18))
}

# to inspect some of the trees:
par(mfrow = c(3, 1))
plot(tree.list[[1]])
plot(tree.list[[4]])
plot(tree.list[[10]])

# estimate sBSDmin for all pairs of trees

sbsdmin <- as.matrix(min.dist.topo.mat(tree.list)[[1]])

groups.example <- get.all.groups(sbsdmin)

#print the partitions

print(groups.example)
```

`convert.to.fasta` *This function is used to convert nexus files to fasta format for clockstar execution*

Description

This function is used to convert nexus files to fasta format for clockstar execution. It uses a folder, and reads all the files with names ending in .nex and converts them to fasta format.

Usage

```
convert.to.fasta(directory)
```

Arguments

`directory`	This is the directory where the files should be located

Details

This function is used internally by run.clockstar() when the files are in nexus format with .nex termination. In this case specify run.clockstar(format = "nex")

Value

This function does not return a value, instead it converts the nexus files in a directory into fasta format.

Note

Internal function used by run.clockstar(). Can also be used directly, see example

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

See user manual for ClockstaR

Examples

```
## Not run:
data(woodmouse)
system("mkdir clockstar_function_example")
write.nexus.data(as.list(woodmouse), file = "./clockstar_function_example/woodmouse.nex")
convert.to.fasta("./clockstar_function_example")

## End(Not run)
## The function is currently defined as
function(directory){
d <- getwd()
setwd(directory)
files <- grep(".nex", dir( ), value=T)
for(i in 1:length(files)){
file.temp <- read.nexus.data(files[i])
write.dna(as.DNAbin(file.temp),file=paste(substr(files[i], 1,nchar(files[i])-4), ".fasta", sep=""),  format
system(paste("rm", files[i]))
}
setwd(d)
}
```

`cut.trees.beta` *cut.trees.beta calculated the diameter of a dendrogram. If the diameter of the dendrogram is higher than that specified by the user, the dendrogram is cut at its longest edge.*

Description

cut.trees.beta calculated the diameter of a dendrogram. If the diameter of the dendrogram is higher than that specified by the user, the dendrogram is cut at its longest edge. This function is internally called by get.all.groups.k and get.all.groups.

Usage

```
cut.trees.beta(tree, beta = 0.05)
```

Arguments

tree	This is a dendrogram of the sBSDmin distances among trees. It should be an object of class "phylo"
beta	This is the threshold sBSDmin distance. If the diameter of the dendrogram is bellow this value, the dendrogram is cut in two. Otherwise the dendrogram is not cut. The default is 0.05. Note that if the larger than the diameter the dendrogram is not cut.

Details

This function is called internally by get.all.groups.k, get.all.groups, and run.clockstar Please see the user manual for fu

Value

comp1	A list with the sub dendrograms as objects of class "phylo". If the dendrogram is cut, the lenghth of the list is two, otherwise it will have a length of one with the same dendrogram supplied.

Note

Please see the user manual for further details

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

get.all.groups.k, get.all.groups, run.clockstar()

Examples

```
## Not run:
# generate an example dendrogram
set.seed(1234)
bsd.example <- unroot(rtree(10))

par(mfrow = c(1, 3))
plot(bsd.example, "unrooted")
subdendrograms <- cut.trees.beta(bsd.example, beta = 0.5)
plot(subdendrograms[[1]], "unrooted")
plot(subdendrograms[[2]], "unrooted")

## End(Not run)
```

```
## The function is currently defined as
function (tree, beta = 0.05)
{
    if (length(tree$tip.label) >= 3) {
        tree <- unroot(tree)
    }
    tree.diam <- max(cophenetic(tree))
    pruned.trees <- list()
    if (tree.diam > beta & (length(tree$tip.label) > 2)) {
        di.nodes <- dist.nodes(tree)
        max.edge <- max(tree$edge.length)
        tips <- 1:length(tree$tip.label)
        nodes <- 1:nrow(di.nodes)
        nodes <- nodes[-tips]
        connect.longest.edge <- which(di.nodes == max.edge, arr.ind = T)[1,
            ]
        if ((connect.longest.edge[1] %in% nodes) & (connect.longest.edge[2] %in%
            nodes)) {
            taxa.cut1 <- tips(tree, connect.longest.edge[1])
            taxa.cut2 <- tips(tree, connect.longest.edge[2])
            len.tax <- c(length(taxa.cut1), length(taxa.cut2))
            node.cut <- connect.longest.edge[len.tax == min(len.tax)]
            taxa.cut <- tips(tree, node.cut)
        }
        else if (sum(connect.longest.edge %in% nodes) == 1) {
            node.cut <- connect.longest.edge[connect.longest.edge %in%
                nodes]
            taxa.cut <- tree$tip.label[connect.longest.edge[connect.longest.edge %in%
                tips]]
        }
        taxa.prune <- taxa.cut
        if ((length(tree$tip.label) - length(taxa.prune)) >=
            2) {
            subtree1 <- drop.tip(tree, taxa.prune)
            if (length(subtree1$tip.label) >= 3) {
                subtree1 <- unroot(subtree1)
            }
        }
        else {
            subtree1 <- tree$tip.label[!(tree$tip.label %in%
                taxa.prune)]
        }
        if (length(taxa.prune) >= 2) {
            subtree2 <- drop.tip(tree, tree$tip.label[!(tree$tip.label %in%
                taxa.prune)])
            if (length(subtree2$tip.label) >= 3) {
                subtree2 <- unroot(subtree2)
            }
        }
        else {
            subtree2 <- taxa.prune
        }
        pruned.trees[[1]] <- subtree1
        pruned.trees[[2]] <- subtree2
    }
    else if (tree.diam < beta) {
```

```
        pruned.trees[[1]] <- tree
    }
    else {
        pruned.trees[[1]] <- tree$tip.label[1]
        pruned.trees[[2]] <- tree$tip.label[2]
    }
    return(pruned.trees)
}
```

---

diagnostics.clockstar *diagnostics.clockstar*

Description

diagnostics.clockstar is a function to inspect the results from the ClockstaR analysis.

Usage

```
diagnostics.clockstar(groups.obj, save.plots = FALSE, plots.file = "clockstar.diagnostics.pdf",
```

Arguments

groups.obj	This should be the output from calling function get.all.groups. For instance, use: example.data <- get.all.groups(sbsdmin.matrix, pam.results = T) example.diagnostics <- diagnostics.clockstar(example.data) sbsdmin.matrix is the matrix of the pairwise sBSDmin matrix estimated in ClockstaR.
save.plots	Should the diagnostic plots should be saved to a pdf file? If so, select T.
plots.file	The name for the pdf file with the plots.
interactive	If this argument is TRUE then ClockstaR will print the diagnostic plots in the active graphics device.

Details

This function can be used to assess the partitioning strategies for different values of k. Please see the tutorial for an example.

Value

comp1	This function prints the Gap statistic for different values of k, and an image matrix with colours to represent the partition assignment.

Note

Please see the user manual for further details

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

get.all.groups.k, get.all.groups, run.clockstar()

Examples

```
## Not run:
# Simulate 10 data sets with three different clocks
fixed.topology <- rtree(10)
fixed.topology$edge.length <- NULL
tree.list <- list()
set.seed(1234)
patterns1 <- abs(rnorm(18, 0, 10))
set.seed(3456)
patterns2 <- abs(rnorm(18, 0, 10))
set.seed(0987)
patterns3 <- abs(rnorm(18, 0, 10))
for(i in 1:3){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns1 + rnorm(18))
}

for(i in 4:7){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns2 + rnorm(18))
}

for(i in 8:10){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns3 + rnorm(18))
}

bsd.matrix.list <- as.matrix(min.dist.topo.mat(tree.list)[[1]])

example.partitions <- get.all.groups(bsd.matrix.list, pam.results = TRUE)

example.diagnostics <- diagnostics.clockstar(example.partitions)

## End(Not run)
## The function is currently defined as
function(groups.obj, save.plots = FALSE, plots.file = "clockstar.diagnostics.pdf", interactive = TRUE)
{
    if(length(groups.obj) == 3){
        stop("The object specified is not the correct format. Please provide an the output of function get.
    }
    gap.stat <- groups.obj[[2]][[1]][, 3]
    ks <- 1:length(gap.stat)
    errs.up <- gap.stat + groups.obj[[2]][[1]][, 4]
    errs.down <- gap.stat - groups.obj[[2]][[1]][, 4]
    k.matrix <- matrix(NA, nrow = nrow(groups.obj[[4]]), ncol = 1)
    for(i in 1:length(ks)){
        part <- pam(groups.obj[[4]], k = ks[i])
        k.matrix <- cbind(k.matrix, as.matrix(part[[3]]))
```

```
    }
    k.matrix <- k.matrix[, 2:ncol(k.matrix)]
    plot.matrices <- function(clust.matrix){
        par(mar = c(2, 4, 2, 4))
        image(t(as.matrix(clust.matrix)), axes = F, col = sample(rainbow(ncol(clust.matrix)*100), ncol(clus
        mtext(text = rownames(clust.matrix), side = 2, line = 0.3, at = seq(0, 1, 1/(nrow(clust.matrix) - 1
        mtext(text = paste0("k=", ks), side = 3, line = 0.3, at = seq(0, 1, 1/(ncol(clust.matrix) - 1)), la
    }
    if(interactive == TRUE){
     par(mfrow = c(2, 1))
     par(mar = c(4, 4, 4, 4))
     errbar(x = ks, y = gap.stat, yplus = errs.up, yminus = errs.down, type = "b", pch = 20, xlab = express
     points(1:length(groups.obj[[3]]), groups.obj[[3]], col = "red", pch = 20)
     lines(1:length(groups.obj[[3]]), groups.obj[[3]], col = "red", pch = 20)
     lines(1:(length(groups.obj[[3]]) + 1), y = rep(0, length(groups.obj[[3]]) + 1), col = "blue")
     legend(x = max(ks)*0.7, y = min(gap.stat)*0.5, legend = c("Gap statistic and SE" ,"Delta Gap statistic
     plot.matrices(k.matrix)
    }
    if(save.plots == TRUE){
        pdf(file = plots.file)
     par(mfrow = c(2, 1))
        par(mar = c(4, 4, 4, 4))
        errbar(x = ks, y = gap.stat, yplus = errs.up, yminus = errs.down, type = "b", pch = 20, xlab = expr
        points(1:length(groups.obj[[3]]), groups.obj[[3]], col = "red", pch = 20)
        lines(1:length(groups.obj[[3]]), groups.obj[[3]], col = "red", pch = 20)
        lines(1:(length(groups.obj[[3]]) + 1), y = rep(0, length(groups.obj[[3]]) + 1), , col = "blue")
        legend(x = max(ks)*0.7, y = min(gap.stat)*0.5, legend = c("Gap statistic and SE" ,"Delta Gap statis
        plot.matrices(k.matrix)
        dev.off()
    }
    return(k.matrix)
}
```

`get.all.groups` *This function finds the best number of partitions (k) for a data set, and the data subsets in each partitions, as described in Duchene et al.*

Description

This function uses the GAP statistic with the PAM algorithm. The input is a matrix or a data frame with the sBSDmin distance between all pairs of trees in the data set.

Usage

```
get.all.groups(data.obj, n.b = 500, pam.results = F, save.partitions = F, file.name = "partition
```

Arguments

`data.obj`	This is the data object. It can be the sBSDmin distance among all pairs of trees, or a dendrogram constructed using these distances. If the data object is the sBSDmin distances, then it should correspond to a data.frame or a matrix object. In this case the partitions are defined using the GAP statistic with the PAM algorithm as implemented in the cluster package.

	If the data object is a dendrogram, the function cuts it along its longest edge, producing two dendrograms. The process is repeated until the diameter of all dendrograms is < a threshold value (known as beta). Note that this method is deprecated as the PAM as the choice of beta is subjective. Only use for test purposes.
`n.b`	Number of bootstrap replicates to estimate the GAP statistic.
`pam.results`	Select T to obtain the results from the GAP statistics. This would be a data frame as estimated in the cluster package. For more details use ?clusGap. If pam.results=T, the function returns a list where the first element is a list of the partitions, and the second is the data frame with the results from clusGap. The default is F.
`save.partitions`	To save the partitions in a text file select T. The default is F
`file.name`	This is the name for the output file with the partitions. This parameter is only used when save.partitions = T.
`beta`	In most analyses this parameter will not be used. This is the threshold value for cutting the dendrogram successively, as described in the above section for data.obj. Please note that this is deprecated, and should only be used for test purposes.
`...`	Additional arguments for clusGap.

Details

See Duchene et al. and the tutorial for more details.

Value

A list where each item is the data subsets in the partition. Partitions are named "partition_1" through "partition_k"

Note

get.all.groups.k is similar to get.all.groups but the user can select the number of partitions (k) instead of the the automatic selection with the GAP statistic implemented here. See user manual and Duchene et al. for futher details

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

Use help(get.all.groups.k) for a version of the function where the number of partitions can be selected.

Examples

```
# This is the same example in the package description
# generate a list of trees with three patterns of among-lineage rate variation
## Not run:

fixed.topology <- rtree(10)
fixed.topology$edge.length <- NULL
tree.list <- list()
set.seed(1234)
patterns1 <- abs(rnorm(18, 0, 10))
set.seed(3456)
patterns2 <- abs(rnorm(18, 0, 10))
set.seed(0987)
patterns3 <- abs(rnorm(18, 0, 10))
for(i in 1:3){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns1 + rnorm(18))
}

for(i in 4:7){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns2 + rnorm(18))
}

for(i in 8:10){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns3 + rnorm(18))
}

# to inspect some of the trees:
par(mfrow = c(3, 1))
plot(tree.list[[1]])
plot(tree.list[[4]])
plot(tree.list[[10]])

# estimate sBSDmin for all pairs of trees

sbsdmin <- as.matrix(min.dist.topo.mat(tree.list)[[1]])

groups.example <- get.all.groups(sbsdmin)

#print the partitions

print(groups.example)

## End(Not run)


## The function is currently defined as
function (data.obj, n.b = 500, pam.results = F, save.partitions = F,
    file.name = "partitions.txt", beta = 0.03, ...)
{
    if (class(data.obj) %in% c("data.frame", "matrix")) {
        kmax <- nrow(data.obj) - 1
        print("FINDING THE BEST NUMBER OF PARTITIONS (k) WITH GAP STATISTIC AND THE PAM ALGORITHM")
        print(kmax)
```

```
    pam.gap <- clusGap(data.obj, pam, B = n.b, K.max = kmax)
    rownames(pam.gap[[1]]) <- paste0("k=", 1:nrow(pam.gap[[1]]))
    gaps <- pam.gap$Tab[, 3]
    diffs <- vector()
    for (i in 1:(length(gaps) - 1)) {
        diffs <- c(diffs, gaps[i] - gaps[i + 1])
    }
    names(diffs) <- paste0("k=", 1:length(diffs))
    all.k <- (1:length(diffs))[diffs > 0]
    best.k <- maxSE(pam.gap[[1]][, 3], pam.gap[[1]][, 4])
    pam.k <- pam(data.obj, k = best.k, ...)
    partitions <- unique(pam.k$clustering)
    res.data <- list()
    for (i in 1:length(partitions)) {
        res.data[[i]] <- names(which(pam.k$clustering ==
            partitions[i]))
    }
    names(res.data) <- paste("Partition_", 1:length(res.data))
    if (save.partitions == T) {
        cat("Partitions selected with automatic mode \n",
            file = file.name)
        for (m in 1:length(res.data)) {
            cat(names(res.data[m]), file = file.name, sep = "\n",
              append = T)
            cat(res.data[[m]], file = file.name, append = T)
            cat("\n", file = file.name, append = T)
        }
    }
    if (pam.results == T) {
        lis.res <- list(res.data, pam.gap, diffs, data.obj)
        return(lis.res)
    }
    else if (pam.results == F) {
        return(res.data)
    }
}
else if (class(data.obj) == "phylo") {
    tree <- data.obj
    temp.list <- list()
    min.list <- list()
    temp.list[[1]] <- tree
    get.diameter <- function(tr) {
        if (class(tr) == "phylo") {
            return(max(cophenetic(tr)))
        }
        else {
            return(0)
        }
    }
    diams <- sapply(temp.list, get.diameter)
    if (any(diams > beta)) {
        while (length(temp.list) != 0) {
            diams <- sapply(temp.list, get.diameter)
            if (sum(diams <= beta) > 0) {
              diams.beta <- seq(from = 1, to = length(diams))[diams <=
                beta]
              for (i in diams.beta) {
```

```
                    min.list[[length(min.list) + 1]] <- temp.list[[i]]
                  }
                }
                else {
                  diams.beta = 0
                }
                if (sum(diams > beta) > 0) {
                  diams.non.beta <- seq(from = 1, to = length(diams))[diams >
                    beta]
                  temp.list.non.beta <- list()
                  for (j in 1:length(diams.non.beta)) {
                    temp.list.non.beta[[j]] <- temp.list[[diams.non.beta[j]]]
                  }
                  temp.list <- temp.list.non.beta
                  sub.list <- list()
                  for (k in 1:length(temp.list)) {
                    cut.temp <- cut.trees.beta(temp.list[[k]],
                      beta)
                    sub.list[[length(sub.list) + 1]] <- cut.temp[[1]]
                    if (length(cut.temp) == 2) {
                      sub.list[[length(sub.list) + 1]] <- cut.temp[[2]]
                    }
                  }
                  temp.list = sub.list[1:length(sub.list)]
                }
                else {
                  temp.list <- list()
                }
            }
        }
        else {
            min.list <- temp.list
        }
        for (l in 1:length(min.list)) {
            if (class(min.list[[l]]) == "phylo") {
                tips <- min.list[[l]]$tip.label
                min.list[[l]] <- tips
            }
        }
        names(min.list) <- paste("Partition_", 1:length(min.list))
        if (save.partitions == T) {
            cat(paste("partitions with selected beta =", beta,
                "\n"), file = file.name)
            for (m in 1:length(min.list)) {
                cat(names(min.list[m]), file = file.name, sep = "\n",
                  append = T)
                cat(min.list[[m]], file = file.name, append = T)
                cat("\n", file = file.name, append = T)
            }
        }
        return(min.list)
    }
    else {
        stop("Please supply the distance between trees. This should be an object of classs Matrix of data.f
    }
}
```

`get.all.groups.k` *This function finds the best partitioning strategy for a fixed number of partitions, as described in Duchene et al.*

Description

This function uses the PAM algorithm to assign data subsets to a fixed number of partitions (k). The input is a matrix or a data frame with the sBSDmin distance between all pairs of trees in the data set. If an object of dendrogram of sBSDmin distances (class phylo) is specified, instead of a matrix or data frame, the function cuts the dendrogram at its longest edge, producing two dendrograms. The process is repeated until a k number of groups is obtained. This function is used when clockstar is run in k mode.

Usage

```
get.all.groups.k(data.obj, k.man = 1, save.partitions = F, file.name = "partitions.txt")
```

Arguments

`data.obj`	This is the data object. It can be the sBSDmin distance among all pairs of trees, or a dendrogram constructed using these distances. If the data object is the sBSDmin distances, then it should correspond to a data.frame or a matrix object. In this case the partitions are defined using the PAM algorithm as implemented in the cluster package. If the data object is a dendrogram, the function cuts it along its longest edge, producing two dendrograms. The process is repeated until the diameter of all dendrograms is the selected k, as specified in the parameter k.man.
`k.man`	The selected number of partitions
`save.partitions`	To save the partitions in a text file select T. The default is F
`file.name`	This is the name for the output file with the partitions. This parameter is only used when save.partitions = T.

Details

See Duchene et al. and the user manual for more details

Value

A list where each item is the data subsets in the partition. Partitions are named "partition_1" through "partition_k"

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

get.all.groups is similar to get.all.groups.k but in get.all.groups the number of partitions is selected automatically using the GAP statistic and the PAM algorithm as implemented in the cluster packages. See user manual and Duchene et al. for futher details

Examples

```
# This is the same example in the package description
# generate a list of trees with three patterns of among-lineage rate variation
## Not run:
fixed.topology <- rtree(10)
fixed.topology$edge.length <- NULL
tree.list <- list()
set.seed(1234)
patterns1 <- abs(rnorm(18, 0, 10))
set.seed(3456)
patterns2 <- abs(rnorm(18, 0, 10))
set.seed(0987)
patterns3 <- abs(rnorm(18, 0, 10))
for(i in 1:3){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns1 + rnorm(18))
}

for(i in 4:7){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns2 + rnorm(18))
}

for(i in 8:10){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns3 + rnorm(18))
}

# to inspect some of the trees:
par(mfrow = c(3, 1))
plot(tree.list[[1]])
plot(tree.list[[4]])
plot(tree.list[[10]])

# estimate sBSDmin for all pairs of trees

sbsdmin <- as.matrix(min.dist.topo.mat(tree.list)[[1]])

groups.example <- get.all.groups.k(sbsdmin, k.man = 3)

#print the partitions

print(groups.example)


## End(Not run)
## The function is currently defined as
function (data.obj, k.man = 1, save.partitions = F, file.name = "partitions.txt")
{
    if (class(data.obj) %in% c("data.frame", "matrix")) {
```

```
    print("The data object is a data frame or a matrix. The partitions will be defined with the PAM alg
    if (k.man == 1) {
        stop("The selected number of partitions is 1. Please select k.man for  >1")
    }
    else {
        pam.k <- pam(data.obj, k = k.man)
        partitions <- unique(pam.k$clustering)
        res.data <- list()
        for (i in 1:length(partitions)) {
            res.data[[i]] <- names(which(pam.k$clustering ==
              partitions[i]))
        }
        names(res.data) <- paste("Partition_", 1:length(res.data))
        if (save.partitions == T) {
            cat(paste("Partitions selected with manual mode for k =",
              k.man, "\n"), file = file.name)
            for (m in 1:length(res.data)) {
              cat(names(res.data[m]), file = file.name, sep = "\n",
                append = T)
              cat(res.data[[m]], file = file.name, append = T)
              cat("\n", file = file.name, append = T)
            }
        }
        return(res.data)
    }
}
else if (class(data.obj) == "phylo") {
    print("The data object is a dendrogram (phylo object). The partitions will be defined but cutting t
    tree <- data.obj
    k = 2
    tree.list <- list()
    tree.list[[1]] <- tree
    while (length(tree.list) < k) {
        diams <- sapply(tree.list, function(tr) {
            if (class(tr) == "phylo") {
              return(max(cophenetic(tr)))
            }
            else {
              return(0)
            }
        })
        tree.to.cut <- tree.list[[which(diams == max(diams))]]
        tree.list <- tree.list[-which(diams == max(diams))]
        tree.list[c(length(tree.list) + 1, length(tree.list) +
            2)] <- cut.trees.beta(tree.to.cut, beta = 0)
    }
    for (l in 1:length(tree.list)) {
        if (class(tree.list[[l]]) == "phylo") {
            tips <- tree.list[[l]]$tip.label
            tree.list[[l]] <- tips
        }
    }
    names(tree.list) <- paste("Partition_", 1:length(tree.list))
    if (save.partitions == T) {
        cat(paste("partitions with selected k =", k, "\n"),
            file = file.name)
        for (m in 1:length(tree.list)) {
```

```
                cat(names(tree.list[m]), file = file.name, sep = "\n",
                  append = T)
                cat(tree.list[[m]], file = file.name, append = T)
                cat("\n", file = file.name, append = T)
            }
        }
        return(tree.list)
    }
}
```

`min.dist.topo` *This function estimates the sBSDmin value described in Duchene et al.*

Description

The function first obtains the minimum branch distance score between a pair of trees by using univariate optimisation. The trees are then rescaled so that the mean sum of their length is = 0.05.

Usage

```
min.dist.topo(tree1, tree2)
```

Arguments

`tree1`	This is a phylogenetic tree as an object of class 'phylo' with branch lengths.
`tree2`	This object is a phylogenetic tree as an object of class phylo with branch lengths.

Details

In the method described in Duchene et al. the topologies of tree1 and tree2 are identical, and they differ in their branch lengths.

Value

The function returns an object of class 'numeric' with three elements as follows: min.bdi.scaled is the beta value for the pair of trees scaling.factor is the scaling factor so that the mean sum of the tree lengths is 0.05. min.bdi is the BSDmin score proposed by Kuhner and Felsenstein (1994) with modifications by Duchene et al.

Note

Please see the corresponding user manual and paper for details of the implementation (Duchene et al.)

Author(s)

Sebastian Duchene

References

See Duchene et al. and Kuhner and Felsenstein (1994)

See Also

min.dist.topo.mat and min.dist.topo.mat.para estimate the pairwise beta for a list of trees.

Examples

```
## Not run:
# Simulate a tree and store it in two objects
tr1 <- rtree(50)
tr2 <- tr1
# change the branch lengths of one of the trees
tr2$edge.length <- abs(rnorm(length(tr1$edge.length), 0.02, 2))

# plot the trees to verify that their specific are different
par(mfrow = c(1, 2))
plot(tr1, show.tip.label = FALSE, direction = "right")
plot(tr2, show.tip.label = FALSE, direction = "left")

#Estimate the beta
min.dist.topo(tr1, tr2)

## End(Not run)

## 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)
```

```
}
```

`min.dist.topo.mat` *Estimate the pairwise BSDmin for a list of trees*

Description

This function calculates the BSDmin distance for a list of trees. For a parallel version use min.dist.topo.mat.para

Usage

```
min.dist.topo.mat(tree.list)
```

Arguments

`tree.list`	A list containing trees with branch lengths. This can be obtained with the clockstaR function optim.edge.lengths

Details

None

Value

This function returns a list with two objects. The first is a dist object with the BSDmin between all pairs of trees. The second is a matrix with the scaling factor used for all paris of trees

`comp1`	dist object with minBSD distances among trees
`comp2`	matrix with scaling factors among trees

Note

Please see the user manual and corresponding paper for further details

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

min.dist.topo.mat.para Please check the user manual and corresponding paper for further details

Examples

```
## Not run:
trees <- list()
tr <- rtree(10)
for(i in 1:10){
trees[[i]] <- tr
trees[[i]]$edge.length <- abs(rnorm(length(tr$edge.length))) + 0.01 # simulate random branch lengths
}

bsd.matrix.list <- min.dist.topo.mat(trees)

par(mfrow = c(2, 1))
hist(bsd.matrix.list[[1]], main = "BSDmin distances in simulated branch lengths", ylab = "Frequency", xlab
hist(bsd.matrix.list[[2]], main = "Scaling factors in simulated branch lengths", ylab = "Frequency", xlab =

## End(Not run)

## 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) {
                min.dist.topo(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 <- min.dist.topo(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
    return(res.list)
  }
```

`min.dist.topo.mat.para` *Estimate the pairwise BSDmin for a list of trees*

Description

This function calculates the BSDmin distance for a list of trees in parallel. It uses the foreach and doParallel packages. It is a faster version than min.dist.topo.mat when the data set is large. But with small data sets, the parallelization may result in slower computation time.

Usage

```
min.dist.topo.mat.para(tree.list, para = F, ncore = 1)
```

Arguments

`tree.list`	A list containing trees with branch lengths. This can be obtained with the clockstaR function optim.edge.lengths
`para`	Select para = T for parallel computation. In such case select the number of cores with ncore. Default is F for no parallelization
`ncore`	Number of cores to run in parallel mode. Default is 1 for no parallelization

Details

None

Value

This function returns a list with two objects. The first is a dist object with the BSDmin between all pairs of trees. The second is a matrix with the scaling factor used for all paris of trees

`comp1`	dist object with minBSD distances among trees
`comp2`	matrix with scaling factors among trees

Note

This function depends on foreach and doParallel packages. Not loading these packages will result in an error. In such case, please use the min.dist.topo.mat function instead. Please check the user manual and corresponding paper for further details

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

min.dist.topo.mat Please check the user manual and corresponding paper for further details

Examples

```
## Not run:
tr <- rtree(10) # simulate a random tree

trees <- list()
for(i in 1:10){
trees[[i]] <- tr
trees[[i]]$edge.length <- abs(rnorm(length(tr$edge.length))) + 0.01 # simulate random branch lengths
}

bsd.matrix.list <- min.dist.topo.mat.para(trees, para = FALSE, ncore = 1)# set para = T and ncore = number

par(mfrow = c(2, 1))
hist(bsd.matrix.list[[1]], main = "BSDmin distances in simulated branch lengths", ylab = "Frequency", xlab
hist(bsd.matrix.list[[2]], main = "Scaling factors in simulated branch lengths", ylab = "Frequency", xlab =



## End(Not run)


## The function is currently defined as
function (tree.list, para = F, ncore = 1)
{
    if (length(tree.list) <= 3) {
        stop("The number of gene trees is <= 3. ClockstaR requires at least gene 4 trees")
    }
    min.dist.topo <- 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(min.dist.topo(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 = c("ape", "phangorn")) %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 = c("ape", "phangorn")) %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)
    return(res.list)
}
```

`optim.edge.lengths` *This function estimates the branch lengths of a fixed tree topology for multiple alignment data sets stored in a folder.*

Description

The function uses the Maximum Likelihood approach implemented in the phangorn package to optimise the branch lengths of a tree for different data sets. When model.test = T the best model is

selected according to the BIC.

Usage

```
optim.edge.lengths(directory, fixed.tree, form = "fasta", model.test = F, save.trees = F, tree.f
```

Arguments

directory	This is the directory with the individual alignments in fasta or nexus format.
fixed.tree	This is the tree for which the branch lengths are estimated. In this function the tree should be an R object. In the run.clockstar version, the tree can be stored in newick format in the folder with the alignments and does not need to be loaded previously in R.
form	The format of the alignments. It can be "nexus", "fasta", or any abbreviation of these. The default is "fasta"
model.test	When mode.test = T, the best fitting substitution model is obtained for each data set using the BIC. Otherwise the branch length optimisation is performed with the Jukes-Cantor model.
save.trees	If the individual trees should be saved, this option should be set to save.trees = T. In this case the trees are written in a newick format. Note that the topologies are identical, but the branch lengths are those of each data set. The trees are saved in the order that the data sets are read into R, which can be verified in the the second item of the list returned by the function. The default is save.trees = F
tree.file.names	The file name for the optimised trees.
para	In multicore machines with the foreach and doParallel packages the function can be run in parallel. Use para = T to enable parallelisation. The default is para = F.
ncore	Number of cores to run in parallel. Only meaningful if para = T

Details

This function is called internally when run.clockstar() is used.

Value

comp1	The first item of the list is a list of trees with the branch lengths optimised for each alignment. Although this is a list, it is defined with class multiPhylo for ease of use with other phylogenetic R packages. The names of the individual trees are those of the data sets.
comp1	The second item of the list is a matrix with the models selected for each data set and their corresponding BIC score. When optim.edge.lengths is used internally from the run.clockstar() function the matrix is saved in the results folder. If para = T, the matrix will be filled with NA, even if model.test = T. Although the parellel version performs model testing, the models selected according to the BIC are not saved.

Note

See the user manual for ClockstaR

Author(s)

Sebastian Duchene

References

Duchene et al.

See Also

run.clockstar, which uses this function internally for a more user friendly implementation.

Examples

```
## Not run:
system("mkdir optim_edge_lengths_test")
tr <- rtree(10)


setwd("./optim_edge_lengths_test")
for(i in 1:10){
tr.temp <- tr
tr.temp$edge.length <- runif(length(tr$edge.length))
data.temp <- as.DNAbin(simSeq(tr))
write.dna(data.temp, file = paste("sim_data", i, ".fasta", sep = ""), format = "fasta")
}
write.tree(tr, file = "sim_fix_tree.tre")

setwd("..")

optim.data <- optim.edge.lengths(directory = "optim_edge_lengths_test", fixed.tree = tr, model.test = TRUE)

par(mfrow = c(2, 5))
par(mar = c(4, 4, 4, 4))
for(i in 1:10) hist(optim.data[[1]][[i]]$edge.length, col = i, main = "", ylab = "Frequency", xlab = paste(

dev.new()
par(mfrow = c(2, 5))
par(mar = c(4, 4, 4, 4))
for(i in 1:10){
plot(optim.data[[1]][[i]])
axisPhylo()
}

print(optim.data[[2]])


## End(Not run)
## The function is currently defined as
function (directory, fixed.tree, form = "fasta", model.test = F,
    save.trees = F, tree.file.names = "output", para = F, ncore = 1)
{
    options(warn = -1)
    directory = paste(directory, "/", sep = "")
    file.names <- dir(directory)
    drop.tip <- ape::drop.tip
    file.names <- file.names[grepl(form, file.names)]
```

```
    model.table <- matrix(NA, nrow = length(file.names), ncol = 3)
    colnames(model.table) <- c("file", "BIC", "model")
    model.table[, 1] <- file.names
    data.files <- list()
    trees.opt <- list()
    print("reading files")
    for (a in 1:length(file.names)) {
        data.files[[a]] <- read.dna(paste(directory, file.names[a],
            sep = ""), format = form)
    }
    if (para == F) {
        for (b in 1:length(file.names)) {
            tax.keep.temp <- fixed.tree$tip.label %in% rownames(data.files[[b]])
            trees.opt[[b]] <- drop.tip(fixed.tree, as.character(fixed.tree$tip.label[!tax.keep.temp]))
            trees.opt[[b]]$edge.length <- rtree(nrow(data.files[[b]]))$edge.length
            pml.temp <- pml(trees.opt[[b]], phyDat(data.files[[b]]),
                inv = 0, shape = 1, k = 1)
            print(paste("model testing dataset", file.names[b],
                b, "of", length(file.names)))
            if (model.test == T) {
             dat.temp <- phyDat(data.files[[b]])
model.temp <- modelTest(dat.temp, multicore = T)
                model.table[b, 2:3] <- c(model.temp$BIC[model.temp$BIC ==
                  min(model.temp$BIC)], model.temp$Model[model.temp$BIC ==
                  min(model.temp$BIC)])
                best.model.temp <- model.temp$Model[model.temp$BIC ==
                  min(model.temp$BIC)][1]
            }
            else if (model.test == F) {
                best.model.temp <- "JC"
            }
            if (length(grep("+G", best.model.temp)) == 0 && length(grep("+I",
                best.model.temp)) == 0) {
                pml.temp <- pml(trees.opt[[b]], phyDat(data.files[[b]]))
                trees.opt[[b]] <- optim.pml(pml.temp, optEdge = T)$tree
            }
            else if (length(grep("+G", best.model.temp)) == 1 &&
                length(grep("+I", best.model.temp)) == 0) {
                pml.temp <- pml(trees.opt[[b]], phyDat(data.files[[b]]),
                  optInv = T)
                trees.opt[[b]] <- optim.pml(pml.temp, optEdge = T,
                  optGamma = T)$tree
            }
            else if (length(grep("+G", best.model.temp)) == 0 &&
                length(grep("+I", best.model.temp)) == 1) {
                pml.temp <- pml(trees.opt[[b]], phyDat(data.files[[b]]),
                  optGamma = T)
                trees.opt[[b]] <- optim.pml(pml.temp, optEdge = T,
                  optInv = T)$tree
            }
            else if (length(grep("+G", best.model.temp)) == 1 &&
                length(grep("+I", best.model.temp)) == 1) {
                pml.temp <- pml(trees.opt[[b]], phyDat(data.files[[b]]),
                  optInv = T, optGamma = T)
                trees.opt[[b]] <- optim.pml(pml.temp, optEdge = T,
                  optGamma = T, optInv = T)$tree
            }
```

```
            print(paste("optimized edge lengths for tree", b,
                "of", length(file.names)))
        }
    }
    optim.trees.par <- function(dat) {
        tree.par <- fixed.tree
        dat.file <- dat
        tax.keep.temp <- fixed.tree$tip.label %in% rownames(dat.file)
        tree.par <- drop.tip(fixed.tree, as.character(fixed.tree$tip.label[!tax.keep.temp]))
        lens.temp <- rtree(nrow(dat.file))
        tree.par$edge.length <- lens.temp$edge.length
        pml.temp <- pml(tree.par, phyDat(dat.file), inv = 0,
            shape = 1, k = 1)
        if (model.test == T) {
         dat.temp <- phyDat(dat)
model.temp <- modelTest(dat.temp, multicore = T)
            best.model.temp <- model.temp$Model[model.temp$BIC ==
                min(model.temp$BIC)][1]
        }
        else if (model.test == F) {
            best.model.temp <- "JC"
        }
        if (length(grep("+G", best.model.temp)) == 0 && length(grep("+I",
            best.model.temp)) == 0) {
            pml.temp <- pml(tree.par, phyDat(dat.file))
            tree.par <- optim.pml(pml.temp, optEdge = T)$tree
        }
        else if (length(grep("+G", best.model.temp)) == 1 &&
            length(grep("+I", best.model.temp)) == 0) {
            pml.temp <- pml(tree.par, phyDat(dat.file), optInv = T)
            tree.par <- optim.pml(pml.temp, optEdge = T, optGamma = T)$tree
        }
        else if (length(grep("+G", best.model.temp)) == 0 &&
            length(grep("+I", best.model.temp)) == 1) {
            pml.temp <- pml(tree.par, phyDat(dat.file), optGamma = T)
            tree.par <- optim.pml(pml.temp, optEdge = T, optInv = T)$tree
        }
        else if (length(grep("+G", best.model.temp)) == 1 &&
            length(grep("+I", best.model.temp)) == 1) {
            pml.temp <- pml(tree.par, phyDat(dat.file), optInv = T,
                optGamma = T)
            tree.par <- optim.pml(pml.temp, optEdge = T, optGamma = T,
                optInv = T)$tree
        }
        return(tree.par)
    }
    if (para == T) {
        print("running parallel version, please wait")
        require(doParallel)
        require(foreach)
        print("making clusters")
        cl <- makeCluster(ncore)
        registerDoParallel(cl)
        print("clusters registered")
        i = data.files
        trees.opt <- foreach(dat = data.files, i = 1:length(data.files),
            .packages = c("phangorn", "ape")) %dopar% optim.trees.par(dat)
```

```
        stopCluster(cl)
        print("parallelized run complete")
    }
    options(warn = 1)
    for (i in 1:length(trees.opt)) {
        names(trees.opt)[i] <- substr(file.names[i], 1, nchar(file.names[i]) -
            nchar(form) - 1)
    }
    if (save.trees == T) {
        print("saving trees")
        class(trees.opt) <- "multiPhylo"
write.tree(trees.opt, file = paste(tree.file.names, ".trees", sep=""), tree.names=T)
    }
    class(trees.opt) <- "multiPhylo"
    l.res <- list()
    l.res[[1]] <- trees.opt
    l.res[[2]] <- model.table
    return(l.res)
  }
```

`run.clockstar` *This function is wrapper for the individual ClockstaR functions. It uses a folder that contains the individual gene alignments and the phylogenetic tree to calculate the BSDmin distance.*

Description

The folder is assigned through a pop up window. It should contain the individual gene alignments in fasta or nexus format and a phylogenetic tree for the relationships among the taxa. The taxon names should be the same in all alignments and in the tree. The output is a folder with a pdf file with some useful plots, two matrices for the pairwise BSDmin distances and the scaling factors for the trees, a dendrogram for the BSDmin distances among genes, the substitution models selected, and the individual gene trees.

Usage

```
run.clockstar(files.directory = "", out.file.name = "", para = F, ncore = 1, model.test = F, moo
```

Arguments

`files.directory`	This is the name of the input folder. This folder should contain the sequence alignment files in fasta or nexus format, and a tree file in newick format.
`out.file.name`	This is the name of the output file, where the results will be stored.
`para`	Use para = T to run the program in parallel. Packages foreach and doParallel are required. When run in parallel with model.test = T clockstar will internally test the substitution model and estimate the branch lengths accordingly. However, a table for the models and BIC scores is not printed.
`ncore`	ncore is used to select the number of cores when the program is run in parallel.
`model.test`	To test the substitution model for each alignment select model.test = T. Default is model.test = F.

mode.run	This option allows the user to select the number of groups 'a priori' using mode.run = "k", or using a threshold for the BSDmin distance among trees with mode.run = "beta".
k.man	If mode.run == "m" select the number of groups. The default is k.man = 2
k.max	If mode.run == "a" select the maximum number of groups. The default is the number of data subsets - 1.
...	Other arguments passed to other functions

Details

This function can be called with no arguments as run.clockstar(). In this case the user is prompted to select the input and output folders, and the values for k and beta.

Value

This function outputs a folder with several files: bsd.plots.pdf has plots for the BSDmin histogram, a coloured plot representing the partitions, the sBSDmin dendrogram.

Note

contact at clockstar website

Author(s)

Sebastian Duchene

References

Duchene et al. ClockstaR: Choosing the number of relaxed-clock models in molecular phylogenetic analysis

See Also

See user manual available at: clockstar website

Examples

```
###############
## Not run:
# create folder to save some simulated data and set the working directory
initial.wd <- getwd()
system("mkdir clockstar_example")
setwd("clockstar_example")

#simulate trees with three different patterns of among-lineage rate variation
# see the example for the function get.all.groups for more details on these simulations.
fixed.topology <- rtree(10)
fixed.topology$edge.length <- NULL
tree.list <- list()
set.seed(1234)
patterns1 <- abs(rnorm(18, 0, 0.5))
set.seed(3456)
patterns2 <- abs(rnorm(18, 0, 0.5))
set.seed(0987)
patterns3 <- abs(rnorm(18, 0, 0.5))
```

```
for(i in 1:3){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns1 + rnorm(18, 0, 0.05))
}

for(i in 4:7){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns2 + rnorm(18, 0, 0.05))
}

for(i in 8:10){
tree.list[[i]] <- fixed.topology
tree.list[[i]]$edge.length <- abs(patterns3 + rnorm(18, 0, 0.05))
}


# Simulate sequence data along each gene tree
secs.data <- list()
for(i in 1:10){
    secs.data[[i]] <- as.DNAbin(simSeq(tree.list[[i]], l = 2000))
}

# Write tree and data to the folder created above
write.tree(fixed.topology, file = "simulation.tre")

for(i in 1:10){
    write.dna(secs.data[[i]], format = "fasta", nbcol = -1, colsep = "", file = paste("secs", i, ".fasta",
}
###############
# Run clockstar using the simulated data in manual mode
#run.clockstar.mod(files.directory = "", out.file.name = "", para = F, ncore = 1, model.test = F, mode.run

run.clockstar(files.directory = getwd(), out.file.name = "clockstar_test_out", mode.run = "a")

# See the results in the "clockstar_test_out" folder
setwd(initial.wd)


## End(Not run)

## The function is currently defined as
run.clockstar <-
function(files.directory = "", out.file.name = "", para = F, ncore = 1, model.test = F, mode.run = "", k.ma
while(files.directory == ""){
print("Please select the file with the sequence data and the phylogenetic tree ")
files.directory <- tk_choose.dir(caption = "Select the folder with the data")
}
if (length(grep("tre|fasta|nex*", dir(files.directory))) == 0){
stop("There are no tree, fasta, or nexus files in the directory. Please make sure the files have the .tre,
}
if (out.file.name == ""){
out.file.name <- readline("Please give a name for the results file (default is clockstar.output): ")
if(out.file.name == ""){
out.file.name <- "clockstar.output"
}
}
    while (mode.run == "") {
```

```
    mode.run <- readline("Run clockstar in automatic or manual mode (type 'a' or 'm')? ")
    if (mode.run == "m") {
        k.man <- readline("Please input a value for k : ")
        k.man <- as.numeric(k.man)
        while (!is.numeric(k.man)) {
            k.man <- readline("Please input a NUMERIC value for k : ")
        }
        print(paste("you have selected a k of: ", k.man))
    }
    if (mode.run == "a") {
        print("ClockstaR will run in automatic mode")
    }
}
files <- dir(files.directory)
files.format <- files[!(grepl(".tre", files) | grepl(out.file.name,
    files))][1]
files.format <- substr(files.format, regexpr("[.]", files.format)[1] +
    1, nchar(files.format))
if (files.format == "nex") {
    convert.to.fasta(files.directory)
}
init.dir <- getwd()
setwd(files.directory)
if (any(dir() == out.file.name)) {
    delete.files <- NA
    while (!(delete.files %in% c("Y", "y", "N", "n"))) {
        delete.files <- readline("A clockstar run with the same output name already exists in this fold
    }
    if (delete.files %in% c("Y", "y")) {
        if (Sys.info()[1] == "Windows") {
            shell(paste("rm -rf", out.file.name))
            shell(paste("mkdir", out.file.name))
        }
        else {
            system(paste("rm -rf", out.file.name))
            system(paste("mkdir", out.file.name))
        }
        setwd(out.file.name)
    }
    else {
        new.dir <- paste(out.file.name, length(grep(out.file.name,
            dir())) + 1, sep = "")
        if (Sys.info()[1] == "Windows") {
            shell(paste("mkdir ", new.dir, sep = ""))
        }
        else {
            system(paste("mkdir ", new.dir, sep = ""))
        }
        setwd(new.dir)
    }
}
else {
    if (Sys.info()[1] == "Windows") {
        shell(paste("mkdir", out.file.name))
    }
    else {
        system(paste("mkdir", out.file.name))
```

```
    }
    setwd(out.file.name)
}
fix.tree <- read.tree(paste(files.directory, files[grep(".tre",
    files)[1]], sep = "/"))
fix.tree$edge.length <- runif(length(fix.tree$tip.label) *
    2 - 1)
print("OPTIMISING BRANCH LENGTHS")
opt.trees <- optim.edge.lengths(files.directory, fix.tree,
    save.trees = TRUE, model.test = model.test, tree.file.names = "optimized",
    para = para, ncore = ncore)
write.table(opt.trees[[2]], file = "models.csv", sep = ",",
    row.names = FALSE)
opt.trees.only <- opt.trees[[1]]
print("FINISHED OPTIMISING BRANCH LENGTHS")
print("CALCULATING BSD")
if (para == F) {
    bsd.matrix <- min.dist.topo.mat(opt.trees.only)
}
else if (para == T) {
    bsd.matrix <- min.dist.topo.mat.para(opt.trees.only,
        para = T, ncore = ncore)
}
print("FINISHED CALCULATING BSD")
write.table(as.matrix(bsd.matrix[[1]]), file = "bsd.distances.csv",
    sep = ",")
write.table(as.matrix(bsd.matrix[[2]]), file = "scaling.factors.csv",
    sep = ",")
if (ncol(bsd.matrix[[2]]) > 2) {
    bsd.dendrogram <- nj(bsd.matrix[[1]])
    write.tree(bsd.dendrogram, file = "bsd.dendrogram.tre")
}
else if (ncol(bsd.matrix[[2]]) == 2) {
    bsd.matrix[[1]] <- as.matrix(bsd.matrix[[1]])
    bsd.matrix[[1]] <- cbind(c(0.5, 0.5), bsd.matrix[[1]])
    bsd.matrix[[1]] <- rbind(c(0.5, 0.5, 0.5), bsd.matrix[[1]])
    colnames(bsd.matrix[[1]])[1] <- 3
    rownames(bsd.matrix[[1]])[1] <- 3
    bsd.matrix[[1]] <- as.dist(bsd.matrix[[1]])
    bsd.dendrogram <- nj(bsd.matrix[[1]])
    bsd.dendrogram <- drop.tip(bsd.dendrogram, "3")
    write.tree(bsd.dendrogram, file = "bsd.dendrogram.tre")
}
else {
    print("ERROR IN DENDROGRAM OF TREE DISTANCES")
}
bsd.data <- as.matrix(bsd.matrix[[1]])
print("CALCULATING NUMBER OF GROUPS")
if (mode.run == "a") {
    print(paste("running in automatic mode"))
    print(bsd.data)
    parts <- get.all.groups(bsd.data, save.partitions = T, ...)
    diagnostics.output <- diagnostics.clockstar(parts, save.plots = TRUE, interactive = FALSE, ...)
}
else if (mode.run == "m") {
    print(paste("running manual mode with k =", k.man))
    parts <- get.all.groups.k(bsd.data, k = k.man, save.partitions = T)
```

```
    }
    print("FINISHED CALCULATING NUMBER OF GROUPS")
    print("PLOTTING")
    if (mode.run == "a"){
parts.pre <- parts
parts <- parts[[1]]
}
    pdf("bsd.plots.pdf")
    par(mfrow = c(3, 1))
    hist(bsd.data, xlab = expression(italic(sBSDmin)), main = expression(italic(sBSDmin)),
        freq = T)
    k.expe.vec <- vector()
    k.expe.names <- vector()
    for (i in 1:length(parts)) {
        k.expe.vec <- c(k.expe.vec, parts[[i]])
        k.expe.names <- c(k.expe.names, rep(names(parts)[i],
            length(parts[[i]])))
    }
    k.expe.names <- as.numeric(as.factor(k.expe.names))
    k.matrix <- k.expe.names
    names(k.matrix) <- k.expe.vec
    image(t(as.matrix(k.matrix)), axes = F, col = rainbow(length(k.matrix)),
        main = "Partition assignments (Colour represents individual partitions)")
    mtext(text = names(k.matrix), side = 2, line = 0.3, at = seq(0,
        1, 1/(length(k.matrix) - 1)), las = 1, cex = 0.8)
    plot(bsd.dendrogram, "unrooted")
    dev.off()

    if (mode.run=="m") {
parts.pre <- get.all.groups(bsd.data, pam.results = TRUE, ...)
diagnostics.output <- diagnostics.clockstar(parts.pre, save.plots = TRUE, interactive = FALSE, ...)
} else {
diagnostics.output <- diagnostics.clockstar(parts.pre, save.plots = TRUE, interactive = FALSE, ...)
}

    setwd(init.dir)
    print("FINISHED RUN")
  }
```

Index