Taxonomic assignment of endophytic isolate E14504F

Dan Spakowicz

October 27, 2017

Herein are the methods for the phylogenetic analysis of isolate E14504F.

# DNA isolation and sequencing

E14504F was grown on potato dextrose broth for 14 days, after which the mycelium was harvested and the genomic DNA purified using a DNeasy Plant Mini Kit (Qiagen; Hilden, Germany) according to the manufacturer’s protocols. The ITS, LSU and RPB2 loci were amplified using GoTaq Flexi (Promega; Madison, WI) and standard primers (Table S1). Amplicons were sequenced at the W.M. Keck Facility at Yale University in forward and reverse directions. Consensus sequences were generated and manually curated for consistency using the Staden package v2.0.0b8 (Bonfield, Smith, and Staden 1995) and were deposited in GenBank.

# Database comparisons

DNA sequences were classified by comparison to several curated online databases, recently reviewed by Prakash et al.(2017). Searching the LSU sequence against the SILVA database returned no taxonomic assignments above confidence threshold. Searching the LSU sequence against the Ribosomal Database Project returned 100% confidence in the genus *Thanatephorus*, within the family Ceratobasidiaceae, order Canthrellales. This genus had been recently circumscribed by Gonzalez et al in an effort to clarify the Rhizoctonia, an anamorphic genus found to be polyphyletic by molecular methods. The BOLD Systems database search of the ITS sequence returned 92% similarity to *Sclerotium hydrophilum*, in the order Agaricales. The UNITE database ITS search resulted in equiprobable matches to *Thanetephorus cucumeris* as well as another genus in the Ceratobasidiaceae, *Ceratorhiza hydrophila*. To clarify the discrepancy between these databases, representatives from each of the clades were retrieved for single multi-locus phylogenetic analysis.

# Phylogenetic analysis

A phylogenetic tree showing the relationship of E14504F to other fungi was constructed using internal transcribed spacer (ITS), large ribosomal subunit (LSU) and RNA polymerase II (RPB2) sequences (Table S2). Comparison sequences were included if voucher specimens of the organism are available in public repositories and the sequence is associated with a publication. Sequences were aligned using the muscle implementaion within the ape package v4.1 (Paradis, Claude, and Strimmer 2004) using R v3.4.2 (R Foundation for Statistical Computing 2017) in RStudio v1.1.383 (Team 2016). Gaps in more than 30% of the alignments were removed. Optimal substitution models were identified for each alignment using the modelTest implentation in the phangorn package v2.2.0 (Schliep 2010). For each locus, a neighbor-joining tree was optimized by maximum likelihood method using a GTR+G model and then 100 bootstrap replicates performed. The loci were then interleaved and a tree contructed using RAxML v8.2.10 (**???**) and visualized using ggtree v1.8.2 (Yu et al. 2017). All sequences used in the study, as well as the individual locus and interleaved trees are available through TreeBase (treebase.org) XXXXXXX. All code for generating the trees, including retrieving comparison sequences, is available at <https://github.com/dspak/e14504f>.

# Code

files <- list.files(path = "../E14504F sequences/",   
 pattern = "\*.fasta", full.names = TRUE)  
ofastas <- list()  
for (f in files) {  
 tmp <- readLines(f)  
 tmp[2:length(tmp)] <- tolower(tmp[2:length(tmp)])  
 ofastas[[f]] <- tmp  
}  
  
# Adjust names of list  
CleanPaths <- function(names){  
 tmp <- gsub(".\*E14504F? (.\*)\\.fasta", "\\1", names)  
 tmp <- gsub(" partial| FULL| ver2", "\\1", tmp)  
 return(tmp)  
}  
names <- CleanPaths(names(ofastas))  
names(ofastas) <- names  
  
# Collapse sequences to one line header, one line sequence  
CollapseTwoLines <- function(fastaString){  
 if (length(fastaString) > 2) {  
 x <- fastaString  
 y <- x[2:length(x)]  
 y <- paste(y, collapse = "")  
 z <- c(x[1], y)  
 return(z)  
 } else {  
 return(fastaString)  
 }  
}  
# Iterate function over the list  
for (i in names(ofastas)) {  
 ofastas[[i]] <- CollapseTwoLines(ofastas[[i]])  
}  
  
# Remove any non-ACTGN nucleotides  
ofastas <- lapply(ofastas, function(x) gsub("^[^>](.\*)[^actgn]", "\\1n", x))

# Load table into dataframe  
sheet <- gs\_title("E14504F")  
x <- gs\_read(sheet)  
  
# Convert hyphen-only columns to NA  
x <- data.frame(apply(x, 2, function(x) gsub("^-$", NA, x)), as.is = TRUE)  
  
# Take in a character vector of genbank accession numbers and return a fasta   
# file in ape format  
RetrieveSequencesAsString <- function(charvec){  
 try({  
 string <- entrez\_fetch(db = "nucleotide", id = charvec, rettype = "fasta")  
 string <- unlist(strsplit(string, split = "\n"))  
 return(string)  
 }, silent = TRUE)  
}  
  
# Retrieve all sequence into a list  
loci <- colnames(x[,6:12])  
gfastas <- list()  
for (i in loci) {  
 gfastas[[i]] <- RetrieveSequencesAsString(x[,grep(i, colnames(x))])  
}  
# Remove those without any sequences  
gfastas <- gfastas[-(which(sapply(gfastas, class) == "try-error"))]  
  
# Convert all to lowercase (needed for nucleotide substitution matrix)  
# and remove any nucleotides other than actg or n  
seqs <- list()  
ConvertToLowercase\_actgn <- function(gfastas) {  
 seqs <- lapply(gfastas, function(x) grep("^[^>]", x))  
 for (i in 1:length(gfastas)) {  
 for (j in 1:length(seqs[[i]])) {  
 tmp <- tolower(gfastas[[i]][seqs[[i]][j]])  
 tmp <- unlist(strsplit(tmp, split = ""))  
 notn <- grep("[^actgn]", tmp)  
 tmp[notn] <- "n"   
 tmp <- paste(tmp, collapse = "")  
 gfastas[[i]][seqs[[i]]][j] <- tmp  
 }  
 }  
 return(gfastas)  
}  
gfastas <- ConvertToLowercase\_actgn(gfastas)

# Create tip labels  
tips <- paste(x$Name, x$Voucher.ID, sep = " ")  
  
# Get names for each of the sequences as list  
hasseq <- apply(x[6:12], 2, function(x) !is.na(x))  
newlabels <- apply(hasseq, 2, function(x) tips[x])  
  
# Reduce labels set to those in gfastas  
newlabels <- newlabels[names(newlabels) %in% names(gfastas)]  
  
# Changes all names in gfastas  
for (i in 1:length(gfastas)) {  
 temp <- tempfile()  
 write(gfastas[[i]], temp)  
 tmp2 <- seqinr::read.fasta(file = temp)  
 temp <- tempfile()  
 seqinr::write.fasta(sequences = tmp2, names = newlabels[[i]], file.out = temp)  
 gfastas[[i]] <- readLines(temp)  
}

# Read in each string to ape as a fasta file  
grfastas <- list()  
for (i in 1:length(gfastas)) {  
 temp <- tempfile()  
 write(gfastas[[i]], temp)  
 grfastas[[i]] <- ape::read.FASTA(file = temp)  
}  
names(grfastas) <- names(gfastas)  
  
# Reverse complement all sequences  
grfastas <- lapply(grfastas, ape::complement)  
  
# Write each to fasta and read back in as string  
for (i in names(grfastas)) {  
 ape::write.dna(x = grfastas[[i]], temp, format = "fasta")  
 grfastas[[i]] <- readLines(temp)  
}

# Create a list for "both fastas" and append the genbank string onto Nneoma's  
bfastas <- list()  
for (n in names(gfastas)){  
 if (n %in% names(ofastas)) {  
 bfastas[[n]] <- c(ofastas[[which(names(ofastas) %in% n)]], gfastas[[n]])  
 } else {  
 bfastas[[n]] <- gfastas[[n]]  
 }  
}  
  
brfastas <- list()  
for (n in names(grfastas)){  
 if (n %in% names(ofastas)) {  
 brfastas[[n]] <- c(ofastas[[which(names(ofastas) %in% n)]], grfastas[[n]])  
 } else {  
 brfastas[[n]] <- grfastas[[n]]  
 }  
}

## Alignments for distance checking

#### ALERT !!!  
# This section takes ~ 10 minutes to run  
# Set to eval=FALSE by default  
#   
# Read in each string as a fasta file  
fastas <- list()  
for (i in 1:length(bfastas)) {  
 temp <- tempfile()  
 write(bfastas[[i]], temp)  
 fastas[[i]] <- ape::read.FASTA(file = temp)  
}  
  
# Align sequences  
alns <- lapply(fastas, ape::muscle)  
names(alns) <- names(bfastas)  
  
# Read in each string as a fasta file  
rfastas <- list()  
for (i in 1:length(brfastas)) {  
 temp <- tempfile()  
 write(brfastas[[i]], temp)  
 rfastas[[i]] <- ape::read.FASTA(file = temp)  
}  
  
# Align sequences  
ralns <- lapply(rfastas, ape::muscle)

# Distance calculation for each alignment  
fdist <- lapply(alns, function(x) dist.dna(x, as.matrix = TRUE))  
rdist <- lapply(ralns, function(x) dist.dna(x, as.matrix = TRUE))  
names(fdist) <- names(rdist) <- names(gfastas)  
  
# Retrieve the E14504F distance (when it's in the sequence)  
compdist <- list()  
for (i in names(fdist)) {  
 # f <- fdist[[i]][,grep("E14504F", colnames(fdist[[i]]))]  
 # r <- rdist[[i]][,grep("E14504F", colnames(rdist[[i]]))]  
 f <- fdist[[i]][,1]  
 r <- rdist[[i]][,1]  
 f <- data.frame(taxon = rownames(fdist[[i]]), forward = f)  
 r <- data.frame(taxon = rownames(rdist[[i]]), revcomp = r)  
 compdist[[i]] <- full\_join(f, r, by = "taxon")  
}  
  
# Plot the results for each alignment  
for (i in names(compdist)) {  
 # Melt each element of the list  
 toplot <- compdist[[i]] %>%  
 gather(key = "direction", value = "distance" , -taxon)  
 # Plot  
 ggplot(toplot, aes(x = taxon, y = distance)) +  
 geom\_point(aes(color = direction)) +  
 labs(x = "Taxon", y = "Distance",  
 title = paste("Forward vs Rev Comp alignments to E14504F: ", i,   
 sep = "")) +  
 theme(axis.text.x = element\_blank())  
 ggsave(paste("indiv\_alignments/directionComp\_", i, ".pdf", sep = ""),   
 height = 4, width = 8)  
}

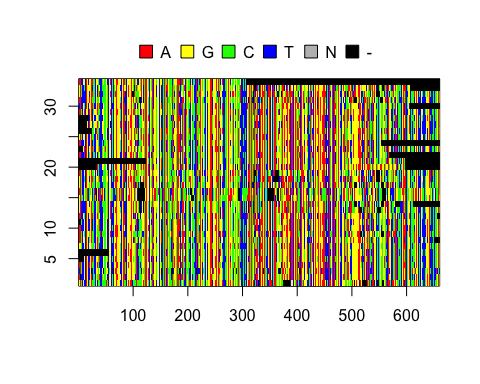
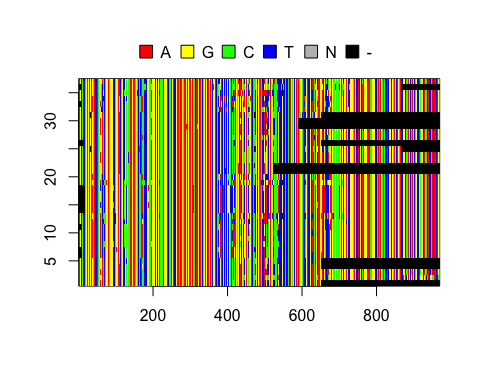
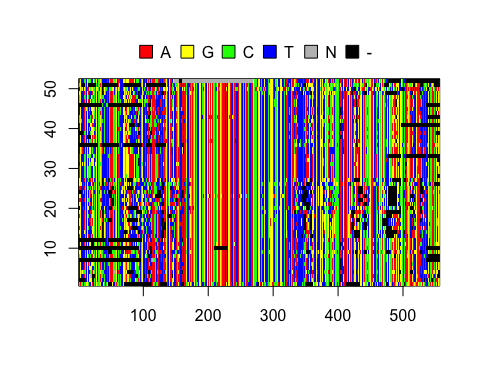
Of the regions checked, only TEF1 contained sequences that had reverse complements. I'll make a tree of just LSU and RPB2. Given the variability observed in the RPB2 region, I expect this will sufficiently resolve the tree.

Update: indeed the resolution was good, but the closest match was to Sclerotium hydrophilium, which does not contain RPB2 and in many cases does not have LSU. All of the Sclerotium in our table contain ITS, though, so I'll make sure to include that region.

# Input regions of interest  
roi <- c("LSU", "RPB2", "ITS")  
roi <- names(alns) %in% roi  
  
# subset alignment  
alns <- alns[roi]

# Remove gaps present in 30% of the columns  
rmgaps <- lapply(alns, function(x) del.colgapsonly(x, threshold = 0.3))  
names(rmgaps) <- names(alns)  
# Write alignments to files for visualization in Jalview  
for (i in names(rmgaps)) {  
 ape::write.dna(rmgaps[[i]], format = "fasta", file = paste("indiv\_alignments/combined\_", i, ".afa", sep = ""))  
}

lapply(rmgaps, function(x) image.DNAbin(x, show.labels = FALSE))



## $ITS  
## $ITS$rect  
## $ITS$rect$w  
## [1] 392.2805  
##   
## $ITS$rect$h  
## [1] 9.62963  
##   
## $ITS$rect$left  
## [1] 82.10976  
##   
## $ITS$rect$top  
## [1] 64.08368  
##   
##   
## $ITS$text  
## $ITS$text$x  
## [1] 126.4715 190.0031 253.5348 317.0665 380.5982 444.1298  
##   
## $ITS$text$y  
## [1] 59.26887 59.26887 59.26887 59.26887 59.26887 59.26887  
##   
##   
##   
## $LSU  
## $LSU$rect  
## $LSU$rect$w  
## [1] 683.6687  
##   
## $LSU$rect$h  
## [1] 6.851852  
##   
## $LSU$rect$left  
## [1] 142.9157  
##   
## $LSU$rect$top  
## [1] 45.76083  
##   
##   
## $LSU$text  
## $LSU$text$x  
## [1] 220.2295 330.9529 441.6762 552.3996 663.1230 773.8463  
##   
## $LSU$text$y  
## [1] 42.33491 42.33491 42.33491 42.33491 42.33491 42.33491  
##   
##   
##   
## $RPB2  
## $RPB2$rect  
## $RPB2$rect$w  
## [1] 465.6567  
##   
## $RPB2$rect$h  
## [1] 6.296296  
##   
## $RPB2$rect$left  
## [1] 97.42166  
##   
## $RPB2$rect$top  
## [1] 42.09626  
##   
##   
## $RPB2$text  
## $RPB2$text$x  
## [1] 150.0812 225.4965 300.9118 376.3271 451.7424 527.1577  
##   
## $RPB2$text$y  
## [1] 38.94811 38.94811 38.94811 38.94811 38.94811 38.94811

# ALERT! This section takes about 10 min to run  
# Get alignment into class pml or phyDat as list inphydat  
inphydat <- list()  
for (i in names(rmgaps)) {  
 temp <- tempfile()  
 ape::write.dna(rmgaps[[i]], format = "fasta", file = temp)  
 inphydat[[i]] <- phangorn::read.phyDat(temp, format = "fasta")  
}  
  
# Check all substitution models for each region  
tests <- lapply(inphydat, function(x) phangorn::modelTest(x, model = "all"))

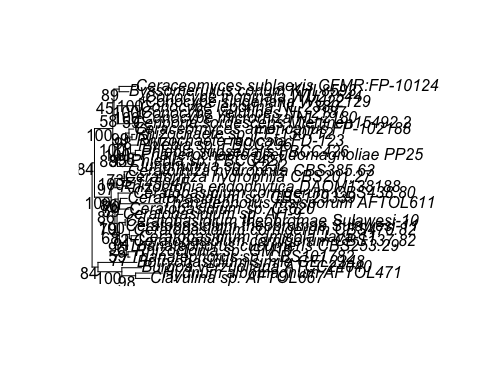
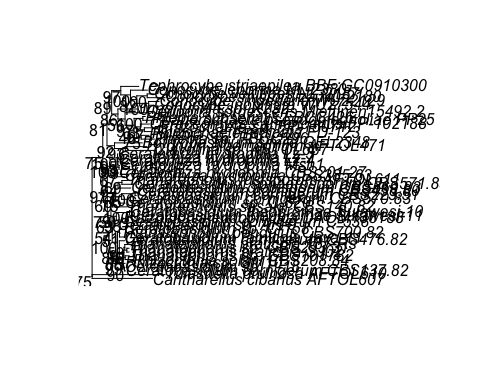
## [1] "JC+I"  
## [1] "JC+G"  
## [1] "JC+G+I"  
## [1] "F81+I"  
## [1] "F81+G"  
## [1] "F81+G+I"  
## [1] "K80+I"  
## [1] "K80+G"  
## [1] "K80+G+I"  
## [1] "HKY+I"  
## [1] "HKY+G"  
## [1] "HKY+G+I"  
## [1] "TrNe+I"  
## [1] "TrNe+G"  
## [1] "TrNe+G+I"  
## [1] "TrN+I"  
## [1] "TrN+G"  
## [1] "TrN+G+I"  
## [1] "TPM1+I"  
## [1] "TPM1+G"  
## [1] "TPM1+G+I"  
## [1] "K81+I"  
## [1] "K81+G"  
## [1] "K81+G+I"  
## [1] "TPM1u+I"  
## [1] "TPM1u+G"  
## [1] "TPM1u+G+I"  
## [1] "TPM2+I"  
## [1] "TPM2+G"  
## [1] "TPM2+G+I"  
## [1] "TPM2u+I"  
## [1] "TPM2u+G"  
## [1] "TPM2u+G+I"  
## [1] "TPM3+I"  
## [1] "TPM3+G"  
## [1] "TPM3+G+I"  
## [1] "TPM3u+I"  
## [1] "TPM3u+G"  
## [1] "TPM3u+G+I"  
## [1] "TIM1e+I"  
## [1] "TIM1e+G"  
## [1] "TIM1e+G+I"  
## [1] "TIM1+I"  
## [1] "TIM1+G"  
## [1] "TIM1+G+I"  
## [1] "TIM2e+I"  
## [1] "TIM2e+G"  
## [1] "TIM2e+G+I"  
## [1] "TIM2+I"  
## [1] "TIM2+G"  
## [1] "TIM2+G+I"  
## [1] "TIM3e+I"  
## [1] "TIM3e+G"  
## [1] "TIM3e+G+I"  
## [1] "TIM3+I"  
## [1] "TIM3+G"  
## [1] "TIM3+G+I"  
## [1] "TVMe+I"  
## [1] "TVMe+G"  
## [1] "TVMe+G+I"  
## [1] "TVM+I"  
## [1] "TVM+G"  
## [1] "TVM+G+I"  
## [1] "SYM+I"  
## [1] "SYM+G"  
## [1] "SYM+G+I"  
## [1] "GTR+I"  
## [1] "GTR+G"  
## [1] "GTR+G+I"  
## [1] "JC+I"  
## [1] "JC+G"  
## [1] "JC+G+I"  
## [1] "F81+I"  
## [1] "F81+G"  
## [1] "F81+G+I"  
## [1] "K80+I"  
## [1] "K80+G"  
## [1] "K80+G+I"  
## [1] "HKY+I"  
## [1] "HKY+G"  
## [1] "HKY+G+I"  
## [1] "TrNe+I"  
## [1] "TrNe+G"  
## [1] "TrNe+G+I"  
## [1] "TrN+I"  
## [1] "TrN+G"  
## [1] "TrN+G+I"  
## [1] "TPM1+I"  
## [1] "TPM1+G"  
## [1] "TPM1+G+I"  
## [1] "K81+I"  
## [1] "K81+G"  
## [1] "K81+G+I"  
## [1] "TPM1u+I"  
## [1] "TPM1u+G"  
## [1] "TPM1u+G+I"  
## [1] "TPM2+I"  
## [1] "TPM2+G"  
## [1] "TPM2+G+I"  
## [1] "TPM2u+I"  
## [1] "TPM2u+G"  
## [1] "TPM2u+G+I"  
## [1] "TPM3+I"  
## [1] "TPM3+G"  
## [1] "TPM3+G+I"  
## [1] "TPM3u+I"  
## [1] "TPM3u+G"  
## [1] "TPM3u+G+I"  
## [1] "TIM1e+I"  
## [1] "TIM1e+G"  
## [1] "TIM1e+G+I"  
## [1] "TIM1+I"  
## [1] "TIM1+G"  
## [1] "TIM1+G+I"  
## [1] "TIM2e+I"  
## [1] "TIM2e+G"  
## [1] "TIM2e+G+I"  
## [1] "TIM2+I"  
## [1] "TIM2+G"  
## [1] "TIM2+G+I"  
## [1] "TIM3e+I"  
## [1] "TIM3e+G"  
## [1] "TIM3e+G+I"  
## [1] "TIM3+I"  
## [1] "TIM3+G"  
## [1] "TIM3+G+I"  
## [1] "TVMe+I"  
## [1] "TVMe+G"  
## [1] "TVMe+G+I"  
## [1] "TVM+I"  
## [1] "TVM+G"  
## [1] "TVM+G+I"  
## [1] "SYM+I"  
## [1] "SYM+G"  
## [1] "SYM+G+I"  
## [1] "GTR+I"  
## [1] "GTR+G"  
## [1] "GTR+G+I"  
## [1] "JC+I"  
## [1] "JC+G"  
## [1] "JC+G+I"  
## [1] "F81+I"  
## [1] "F81+G"  
## [1] "F81+G+I"  
## [1] "K80+I"  
## [1] "K80+G"  
## [1] "K80+G+I"  
## [1] "HKY+I"  
## [1] "HKY+G"  
## [1] "HKY+G+I"  
## [1] "TrNe+I"  
## [1] "TrNe+G"  
## [1] "TrNe+G+I"  
## [1] "TrN+I"  
## [1] "TrN+G"  
## [1] "TrN+G+I"  
## [1] "TPM1+I"  
## [1] "TPM1+G"  
## [1] "TPM1+G+I"  
## [1] "K81+I"  
## [1] "K81+G"  
## [1] "K81+G+I"  
## [1] "TPM1u+I"  
## [1] "TPM1u+G"  
## [1] "TPM1u+G+I"  
## [1] "TPM2+I"  
## [1] "TPM2+G"  
## [1] "TPM2+G+I"  
## [1] "TPM2u+I"  
## [1] "TPM2u+G"  
## [1] "TPM2u+G+I"  
## [1] "TPM3+I"  
## [1] "TPM3+G"  
## [1] "TPM3+G+I"  
## [1] "TPM3u+I"  
## [1] "TPM3u+G"  
## [1] "TPM3u+G+I"  
## [1] "TIM1e+I"  
## [1] "TIM1e+G"  
## [1] "TIM1e+G+I"  
## [1] "TIM1+I"  
## [1] "TIM1+G"  
## [1] "TIM1+G+I"  
## [1] "TIM2e+I"  
## [1] "TIM2e+G"  
## [1] "TIM2e+G+I"  
## [1] "TIM2+I"  
## [1] "TIM2+G"  
## [1] "TIM2+G+I"  
## [1] "TIM3e+I"  
## [1] "TIM3e+G"  
## [1] "TIM3e+G+I"  
## [1] "TIM3+I"  
## [1] "TIM3+G"  
## [1] "TIM3+G+I"  
## [1] "TVMe+I"  
## [1] "TVMe+G"  
## [1] "TVMe+G+I"  
## [1] "TVM+I"  
## [1] "TVM+G"  
## [1] "TVM+G+I"  
## [1] "SYM+I"  
## [1] "SYM+G"  
## [1] "SYM+G+I"  
## [1] "GTR+I"  
## [1] "GTR+G"  
## [1] "GTR+G+I"

# Retrieve the best models  
bestmodel <- sapply(tests, function(x) x$Model[which.max(x$logLik)])  
names(bestmodel) <- names(rmgaps)

# Convert to phyDat format  
tophydat <- lapply(rmgaps, phyDat)  
  
# Create a distance matrix  
onetree <- lapply(tophydat, dist.ml)  
  
# Create neighbor-joining tree  
ot <- lapply(onetree, NJ)  
  
# Maximum Likelihood on the NJ tree with the alignment  
ot.ml <- list()  
for (i in names(ot)) {  
 ot.ml[[i]] <- pml(ot[[i]], tophydat[[i]])  
}  
  
# Apply GTR+G model  
optim.ml <- lapply(ot.ml, function(x) optim.pml(x, model = "GTR",   
 rearrangement = "stochastic",   
 optGamma = TRUE))

## optimize edge weights: -11728.36 --> -11479.92   
## optimize base frequencies: -11479.92 --> -11443.71   
## optimize rate matrix: -11443.71 --> -11235.26   
## optimize edge weights: -11235.26 --> -11233.69   
## optimize topology: -11233.69 --> -11218.83   
## optimize topology: -11218.83 --> -11213.82   
## optimize topology: -11213.82 --> -11211.64   
## 11   
## optimize base frequencies: -11211.64 --> -11208.67   
## optimize rate matrix: -11208.67 --> -11206.85   
## optimize edge weights: -11206.85 --> -11206.82   
## optimize topology: -11206.82 --> -11206.82   
## 0   
## [1] "Ratchet iteration 1 , best pscore so far: -11206.8231972991"  
## [1] "Ratchet iteration 2 , best pscore so far: -11206.8231972991"  
## [1] "Ratchet iteration 3 , best pscore so far: -11206.8231972991"  
## [1] "Ratchet iteration 4 , best pscore so far: -11206.8231972991"  
## [1] "Ratchet iteration 5 , best pscore so far: -11206.8231972991"  
## [1] "Ratchet iteration 6 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 7 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 8 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 9 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 10 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 11 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 12 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 13 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 14 , best pscore so far: -11206.8231966449"  
## [1] "Ratchet iteration 15 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 16 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 17 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 18 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 19 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 20 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 21 , best pscore so far: -11206.8231958623"  
## [1] "Ratchet iteration 22 , best pscore so far: -11201.8435677182"  
## [1] "Ratchet iteration 23 , best pscore so far: -11201.8435677182"  
## [1] "Ratchet iteration 24 , best pscore so far: -11201.8435677182"  
## [1] "Ratchet iteration 25 , best pscore so far: -11201.8435677182"  
## [1] "Ratchet iteration 26 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 27 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 28 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 29 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 30 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 31 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 32 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 33 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 34 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 35 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 36 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 37 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 38 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 39 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 40 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 41 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 42 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 43 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 44 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 45 , best pscore so far: -11201.8435675968"  
## [1] "Ratchet iteration 46 , best pscore so far: -11201.8435675968"  
## optimize base frequencies: -11201.84 --> -11200.76   
## optimize rate matrix: -11200.76 --> -11200.38   
## optimize edge weights: -11200.38 --> -11200.36   
## optimize topology: -11200.36 --> -11200.36   
## 0   
## optimize base frequencies: -11200.36 --> -11200.22   
## optimize rate matrix: -11200.22 --> -11200.17   
## optimize edge weights: -11200.17 --> -11200.17   
## optimize base frequencies: -11200.17 --> -11200.14   
## optimize rate matrix: -11200.14 --> -11200.13   
## optimize edge weights: -11200.13 --> -11200.13   
## optimize base frequencies: -11200.13 --> -11200.13   
## optimize rate matrix: -11200.13 --> -11200.13   
## optimize edge weights: -11200.13 --> -11200.13   
## optimize base frequencies: -11200.13 --> -11200.13   
## optimize rate matrix: -11200.13 --> -11200.12   
## optimize edge weights: -11200.12 --> -11200.12   
## optimize base frequencies: -11200.12 --> -11200.12   
## optimize rate matrix: -11200.12 --> -11200.12   
## optimize edge weights: -11200.12 --> -11200.12   
## optimize base frequencies: -11200.12 --> -11200.12   
## optimize rate matrix: -11200.12 --> -11200.12   
## optimize edge weights: -11200.12 --> -11200.12   
## optimize edge weights: -6116.817 --> -6075.341   
## optimize base frequencies: -6075.341 --> -6071.346   
## optimize rate matrix: -6071.346 --> -5817.837   
## optimize edge weights: -5817.837 --> -5817.274   
## optimize topology: -5817.274 --> -5801.738   
## optimize topology: -5801.738 --> -5793.015   
## optimize topology: -5793.015 --> -5783.298   
## 4   
## optimize base frequencies: -5783.298 --> -5780.7   
## optimize rate matrix: -5780.7 --> -5779.705   
## optimize edge weights: -5779.705 --> -5779.659   
## optimize topology: -5779.659 --> -5771.006   
## optimize topology: -5771.006 --> -5766.97   
## optimize topology: -5766.97 --> -5766.383   
## 4   
## optimize base frequencies: -5766.383 --> -5765.984   
## optimize rate matrix: -5765.984 --> -5765.85   
## optimize edge weights: -5765.85 --> -5765.84   
## optimize topology: -5765.84 --> -5765.657   
## optimize topology: -5765.657 --> -5765.657   
## 1   
## optimize base frequencies: -5765.657 --> -5765.617   
## optimize rate matrix: -5765.617 --> -5765.602   
## optimize edge weights: -5765.602 --> -5765.601   
## optimize topology: -5765.601 --> -5765.601   
## 0   
## [1] "Ratchet iteration 1 , best pscore so far: -5765.60080337579"  
## [1] "Ratchet iteration 2 , best pscore so far: -5765.60080337579"  
## [1] "Ratchet iteration 3 , best pscore so far: -5765.60080337579"  
## [1] "Ratchet iteration 4 , best pscore so far: -5765.60080337579"  
## [1] "Ratchet iteration 5 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 6 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 7 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 8 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 9 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 10 , best pscore so far: -5761.63845200367"  
## [1] "Ratchet iteration 11 , best pscore so far: -5761.63845199346"  
## [1] "Ratchet iteration 12 , best pscore so far: -5756.34303038757"  
## [1] "Ratchet iteration 13 , best pscore so far: -5756.34303038757"  
## [1] "Ratchet iteration 14 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 15 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 16 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 17 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 18 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 19 , best pscore so far: -5756.34303033444"  
## [1] "Ratchet iteration 20 , best pscore so far: -5756.25346593067"  
## [1] "Ratchet iteration 21 , best pscore so far: -5756.25346260705"  
## [1] "Ratchet iteration 22 , best pscore so far: -5756.25346260705"  
## [1] "Ratchet iteration 23 , best pscore so far: -5756.25346260705"  
## [1] "Ratchet iteration 24 , best pscore so far: -5756.25346260705"  
## [1] "Ratchet iteration 25 , best pscore so far: -5756.25346260705"  
## [1] "Ratchet iteration 26 , best pscore so far: -5756.15297177589"  
## [1] "Ratchet iteration 27 , best pscore so far: -5756.15297177589"  
## [1] "Ratchet iteration 28 , best pscore so far: -5756.15297177589"  
## [1] "Ratchet iteration 29 , best pscore so far: -5756.15297177589"  
## [1] "Ratchet iteration 30 , best pscore so far: -5756.15297177589"  
## [1] "Ratchet iteration 31 , best pscore so far: -5756.15297171215"  
## [1] "Ratchet iteration 32 , best pscore so far: -5756.15297171215"  
## [1] "Ratchet iteration 33 , best pscore so far: -5756.15297171215"  
## [1] "Ratchet iteration 34 , best pscore so far: -5756.15297171215"  
## [1] "Ratchet iteration 35 , best pscore so far: -5756.15297171215"  
## [1] "Ratchet iteration 36 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 37 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 38 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 39 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 40 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 41 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 42 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 43 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 44 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 45 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 46 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 47 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 48 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 49 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 50 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 51 , best pscore so far: -5756.15297151381"  
## [1] "Ratchet iteration 52 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 53 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 54 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 55 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 56 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 57 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 58 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 59 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 60 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 61 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 62 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 63 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 64 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 65 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 66 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 67 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 68 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 69 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 70 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 71 , best pscore so far: -5756.15297146189"  
## [1] "Ratchet iteration 72 , best pscore so far: -5756.15297146189"  
## optimize base frequencies: -5756.153 --> -5756.118   
## optimize rate matrix: -5756.118 --> -5756.091   
## optimize edge weights: -5756.091 --> -5756.09   
## optimize topology: -5756.09 --> -5756.09   
## 0   
## optimize base frequencies: -5756.09 --> -5756.089   
## optimize rate matrix: -5756.089 --> -5756.089   
## optimize edge weights: -5756.089 --> -5756.089   
## optimize base frequencies: -5756.089 --> -5756.088   
## optimize rate matrix: -5756.088 --> -5756.088   
## optimize edge weights: -5756.088 --> -5756.088   
## optimize base frequencies: -5756.088 --> -5756.088   
## optimize rate matrix: -5756.088 --> -5756.088   
## optimize edge weights: -5756.088 --> -5756.088   
## optimize edge weights: -12908.75 --> -12758.5   
## optimize base frequencies: -12758.5 --> -12748.82   
## optimize rate matrix: -12748.82 --> -12112.97   
## optimize edge weights: -12112.97 --> -12104.64   
## optimize topology: -12104.64 --> -12082.09   
## optimize topology: -12082.09 --> -12073.9   
## optimize topology: -12073.9 --> -12071.44   
## 8   
## optimize base frequencies: -12071.44 --> -12064.97   
## optimize rate matrix: -12064.97 --> -12060.32   
## optimize edge weights: -12060.32 --> -12060.07   
## optimize topology: -12060.07 --> -12054.43   
## optimize topology: -12054.43 --> -12054.43   
## 1   
## optimize base frequencies: -12054.43 --> -12052.13   
## optimize rate matrix: -12052.13 --> -12051.06   
## optimize edge weights: -12051.06 --> -12050.97   
## optimize topology: -12050.97 --> -12050.97   
## 0   
## [1] "Ratchet iteration 1 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 2 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 3 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 4 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 5 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 6 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 7 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 8 , best pscore so far: -12050.9692203653"  
## [1] "Ratchet iteration 9 , best pscore so far: -12050.6366974563"  
## [1] "Ratchet iteration 10 , best pscore so far: -12050.636692327"  
## [1] "Ratchet iteration 11 , best pscore so far: -12050.636692327"  
## [1] "Ratchet iteration 12 , best pscore so far: -12050.636692327"  
## [1] "Ratchet iteration 13 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 14 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 15 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 16 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 17 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 18 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 19 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 20 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 21 , best pscore so far: -12050.6366922499"  
## [1] "Ratchet iteration 22 , best pscore so far: -12050.6366921879"  
## [1] "Ratchet iteration 23 , best pscore so far: -12050.6366921879"  
## [1] "Ratchet iteration 24 , best pscore so far: -12050.6366921879"  
## [1] "Ratchet iteration 25 , best pscore so far: -12050.6366921879"  
## [1] "Ratchet iteration 26 , best pscore so far: -12050.6366921879"  
## [1] "Ratchet iteration 27 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 28 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 29 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 30 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 31 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 32 , best pscore so far: -12046.8440662567"  
## [1] "Ratchet iteration 33 , best pscore so far: -12046.8440659366"  
## [1] "Ratchet iteration 34 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 35 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 36 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 37 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 38 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 39 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 40 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 41 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 42 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 43 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 44 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 45 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 46 , best pscore so far: -12046.8440658446"  
## [1] "Ratchet iteration 47 , best pscore so far: -12045.5136875533"  
## [1] "Ratchet iteration 48 , best pscore so far: -12045.5136875533"  
## [1] "Ratchet iteration 49 , best pscore so far: -12045.5136875533"  
## [1] "Ratchet iteration 50 , best pscore so far: -12045.5136875533"  
## [1] "Ratchet iteration 51 , best pscore so far: -12045.5136867006"  
## [1] "Ratchet iteration 52 , best pscore so far: -12045.5136867006"  
## [1] "Ratchet iteration 53 , best pscore so far: -12045.5136862503"  
## [1] "Ratchet iteration 54 , best pscore so far: -12045.5136862503"  
## [1] "Ratchet iteration 55 , best pscore so far: -12045.5136862503"  
## [1] "Ratchet iteration 56 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 57 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 58 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 59 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 60 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 61 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 62 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 63 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 64 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 65 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 66 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 67 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 68 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 69 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 70 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 71 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 72 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 73 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 74 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 75 , best pscore so far: -12045.513685698"  
## [1] "Ratchet iteration 76 , best pscore so far: -12045.513685698"  
## optimize base frequencies: -12045.51 --> -12044.85   
## optimize rate matrix: -12044.85 --> -12044.28   
## optimize edge weights: -12044.28 --> -12044.26   
## optimize topology: -12044.26 --> -12044.26   
## 0   
## optimize base frequencies: -12044.26 --> -12044.02   
## optimize rate matrix: -12044.02 --> -12043.89   
## optimize edge weights: -12043.89 --> -12043.88   
## optimize base frequencies: -12043.88 --> -12043.8   
## optimize rate matrix: -12043.8 --> -12043.76   
## optimize edge weights: -12043.76 --> -12043.76   
## optimize base frequencies: -12043.76 --> -12043.74   
## optimize rate matrix: -12043.74 --> -12043.72   
## optimize edge weights: -12043.72 --> -12043.72   
## optimize base frequencies: -12043.72 --> -12043.71   
## optimize rate matrix: -12043.71 --> -12043.71   
## optimize edge weights: -12043.71 --> -12043.71   
## optimize base frequencies: -12043.71 --> -12043.71   
## optimize rate matrix: -12043.71 --> -12043.7   
## optimize edge weights: -12043.7 --> -12043.7   
## optimize base frequencies: -12043.7 --> -12043.7   
## optimize rate matrix: -12043.7 --> -12043.7   
## optimize edge weights: -12043.7 --> -12043.7   
## optimize base frequencies: -12043.7 --> -12043.7   
## optimize rate matrix: -12043.7 --> -12043.7   
## optimize edge weights: -12043.7 --> -12043.7   
## optimize base frequencies: -12043.7 --> -12043.7   
## optimize rate matrix: -12043.7 --> -12043.7   
## optimize edge weights: -12043.7 --> -12043.7   
## optimize base frequencies: -12043.7 --> -12043.7   
## optimize rate matrix: -12043.7 --> -12043.7   
## optimize edge weights: -12043.7 --> -12043.7

# Bootstrap replicates  
bs <- lapply(optim.ml, function(x) bootstrap.pml(x, bs = 100, optNni = TRUE,   
 multicore = TRUE,   
 control = pml.control(trace = 0)))  
# Make into tree plotting file  
trees <- list()  
for (i in names(ot)) {  
 trees[[i]] <- plotBS(optim.ml[[i]]$tree, bs[[i]], p = 20, type = "p")  
}



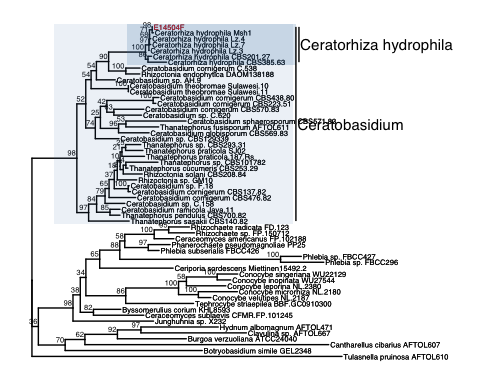
# If root taxa are present, reroot, else make an unrooted tree  
# There's some funky xlimit spacing here - I couldn't figure out a way to make   
# it scale with the branch lengths or something in the tree object  
PlotSingleLocusTree <- function(tree, i) {  
 p <- ggtree(tree) +  
 geom\_tiplab(size = 3) +  
 geom\_text2(aes(subset = !isTip, label = label),   
 size = 2, hjust = 1.1, vjust = -0.5) +  
 labs(title = paste("E14505F single locus tree: ", i, sep = ""))  
 if (sum(tree$edge.length) > 1) {  
 p <- p + xlim\_tree(1.1)  
 } else {  
 p <- p + xlim\_tree(0.3)  
 }  
 p <- p + ggsave(paste("singleLocusTree\_", i, ".pdf", sep = ""),  
 height = 9, width = 6.5)   
}  
  
for (i in names(trees)) {  
 if ("Tulasnella pruinosa AFTOL610" %in% trees[[i]]$tip.label) {  
 tree <- root(trees[[i]], outgroup = "Tulasnella pruinosa AFTOL610",   
 resolve.root = FALSE, edgelabel = TRUE)  
 PlotSingleLocusTree(tree, i)  
 } else {  
 tree <- unroot(trees[[i]])  
 PlotSingleLocusTree(tree, i)  
 }  
}  
  
# save(trees, file = "single-locus-trees.RData")

# Get all names in all alignments  
intnames <- unique(unlist(sapply(rmgaps, rownames)))  
  
# Add a sequence of N's for taxa without that locus  
allns <- list()  
for (i in names(rmgaps)) {  
 # Grab all names NOT in the alignment  
 tmp\_names <- intnames[!(intnames %in% rownames(rmgaps[[i]]))]  
 # Create a list of N's for each name equal to alignment length  
 tmp\_seqs <- vector(mode = "list", length = length(tmp\_names))  
 for (j in 1:length(tmp\_seqs)) {  
 tmp\_seqs[[j]] <- rep("n", ncol(rmgaps[[i]]))  
 }  
 # Convert to DNAbin  
 tmp\_seqs <- ape::as.DNAbin(tmp\_seqs)  
 # Convert to muscle alignment matrix  
 tmp\_seqs <- ape::muscle(tmp\_seqs)  
 rownames(tmp\_seqs) <- tmp\_names  
 # Bind to N sequences to that alignment  
 allns[[i]] <- rbind(rmgaps[[i]], tmp\_seqs)  
}  
  
# Bind all of the alignments into one  
interleaved <- do.call("cbind", allns)

# Adjust names  
rownames(interleaved) <- make.names(rownames(interleaved))

# Create phylip output for raxml  
write.phy(interleaved, "all.phy")  
  
# Write Mr Bayes submission file to move to Grace and run there.  
fileConn <- file("mrbayes\_e14504f\_all.sh")  
writeLines(c("#!/bin/bash",   
 "#SBATCH --ntasks=4 --nodes=1",  
 "#SBATCH --time=12:00:00",  
 "#SBATCH --job-name=mrbayes\_e14504f",  
 "#SBATCH --mail-user=daniel.spakowicz@yale.edu",  
 "#SBATCH --mail-type=ALL",  
 "",  
 "# Produced by DS\_tree.Rmd",  
 "# Dan Spakowicz",  
 paste("# ", Sys.time(), sep = ""),  
 "# Submission file for the MrBayes analysis of the E14504F tree using the sequencing in the google sheet 'E14504F'",  
 "",  
 "cd /project/fas/gerstein/djs88/e14504f",  
 "module load Apps/MrBayes/3.2.2",  
 "mpirun mb all.nex"), fileConn)  
close(fileConn)  
  
# Create a new file for RAxML (find the largest existing file and add one)  
ls <- list.files(pattern = ".\*all\_\\d+")  
maxiter <- max(as.numeric(gsub(".\*all\_(\\d+)", "\\1", ls)))  
  
# Call RAxML (takes ~20 min)  
system({  
 paste("raxmlHPC-AVX -s all.phy -m GTRGAMMA -n all\_", maxiter+1, " -f a -# 100 -x 12345 -p 12345", sep = "")  
})  
system({  
 paste("raxmlHPC-AVX -m GTRGAMMA -J MRE -z RAxML\_bootstrap.all\_", maxiter+1, " -n tre", maxiter+1, sep = "")  
})

t <- read.tree(paste("RAxML\_bipartitions.all\_", maxiter+1, sep = ""))  
  
int <- root(t, outgroup = "Tulasnella.pruinosa.AFTOL610",   
 resolve.root = FALSE, edgelabel = TRUE)  
  
cleanTipLabels <- function(tiplabels) {  
 x <- gsub("([a-z])\\.([a-z|A-Z])", "\\1 \\2", tiplabels)  
 y <- gsub("..", ". ", x, fixed = TRUE)  
 return(y)  
}  
  
int$tip.label <- cleanTipLabels(int$tip.label)  
  
p <- ggtree(int) +  
 geom\_tiplab(size=2) +  
 geom\_text2(aes(subset = !isTip, label = label),   
 size = 2, hjust = 1.1, vjust = -0.5) +  
 xlim\_tree(1.1)  
  
# Create an annotation data frame  
annot <- data.frame(label = int$tip.label,  
 group = "1", stringsAsFactors = FALSE)  
annot[grep("E145", annot$label),] <- c("E14504F", "2")  
annot$group <- as.factor(annot$group)  
  
# Add tip label annotation  
p %<+% annot +   
 geom\_tiplab(aes(color = group), size = 2) +  
 scale\_color\_manual(values = c("black", "red")) +  
 geom\_hilight(76, fill = "steelblue", alpha = 0.1, extend = 0.28) +  
 geom\_hilight(95, fill = "steelblue", alpha = 0.2, extend = 0.34) +  
 geom\_cladelabel(node = 76, label = "Ceratobasidium",   
 offset = 0.28, fontsize = 4) +  
 geom\_cladelabel(node = 95, label = "Ceratorhiza hydrophila",   
 offset = 0.34, fontsize = 4) +  
 ggsave("all\_interleaved.pdf", height = 4.75, width = 7.5)



# References

Bonfield, J K, K f Smith, and R Staden. 1995. “A New DNA Sequence Assembly Program.” *Nucleic Acids Research* 23 (24): 4992–9. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC307504/>.

Paradis, Emmanuel, Julien Claude, and Korbinian Strimmer. 2004. “APE: Analyses of Phylogenetics and Evolution in R Language.” *Bioinformatics* 20 (2): 289–90. doi:[10.1093/bioinformatics/btg412](https://doi.org/10.1093/bioinformatics/btg412).

Prakash, Peralam Yegneswaran, Laszlo Irinyi, Catriona Halliday, Sharon Chen, Vincent Robert, and Wieland Meyer. 2017. “Online Databases for the Taxonomy and Identification of Pathogenic Fungi and Proposal for a Cloud-Based Dynamic Data Network Platform.” *Journal of Clinical Microbiology*, JCM–02084.

R Foundation for Statistical Computing, Austria., Vienna. 2017. “R Core Team (2017). R: A Language and Environment for Statistical Computing.” *Vienna, Austria. Internet: Http://Www. R-Project. Org*. <https://www.R-project.org/>.

Schliep, Klaus Peter. 2010. “Phangorn: Phylogenetic Analysis in R.” *Bioinformatics*, btq706.

Team, RStudio. 2016. “RStudio: Integrated Development Environment for R.” Boston, MA: RStudio, Inc. <https://www.rstudio.com/>.

Yu, Guangchuang, David K. Smith, Huachen Zhu, Yi Guan, and Tommy Tsan-Yuk Lam. 2017. “Ggtree: An R Package for Visualization and Annotation of Phylogenetic Trees with Their Covariates and Other Associated Data.” *Methods in Ecology and Evolution* 8 (1): 28–36.