Chapter 4b ITS DADA2-specific processing step

Emily Giroux

09/08/2020

Using package `BiocManager` to install required packages:

```
r <- getOption("repos")
r["CRAN"] <- "http://cran.us.r-project.org"
options(repos = r)

if (!requireNamespace("BiocManager"))
    install.packages("BiocManager")
BiocManager::install()

library("BiocManager")
.cran_packages <- c("cowplot", "data.table", "ggplot2", "knitr", "rprojroot")
.bioc_packages <- c("BiocStyle", "Biostrings", "dada2", "ShortRead")
.inst <- .cran_packages %in% installed.packages()
if(any(!.inst)) {
   install.packages(.cran_packages[!.inst])
}
.inst <- .bioc_packages %in% installed.packages()
if(any(!.inst)) {
  BiocManager::install(.bioc_packages[!.inst], ask = FALSE)
}
```

Load packages into session, and print package versions:

```
sapply(c(.cran_packages, .bioc_packages), require, character.only = TRUE)
```

```
##    cowplot data.table    ggplot2      knitr  rprojroot  BiocStyle Biostrings
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
##      dada2  ShortRead
##       TRUE       TRUE
```

Load the saved image from Chapter 2, then save it as a separate image to retain environment data specific to the ITS processing and analysis workflow.

```
load(paste(imageDirPath, chptImageA, sep = ""))
chptImageB   <- "ecobiomics_ITS_2b.RData"
save.image(paste(imageDirPath, chptImageA, sep = ""))
```

When re-starting a session, you can quickly load up the image by running the chunk below:

```
sharedPath <- "/isilon/cfia-ottawa-fallowfield/users/girouxeml/PIRL_working_directory/"
analysis <- "ecobiomics/"
sharedPathAn <- paste(sharedPath, analysis, sep = "/")
imageDirPath <- "/home/CFIA-ACIA/girouxeml/GitHub_Repos/r_environments/ecobiomics/"
chptImageB   <- "ecobiomics_ITS_2b.RData"
load(paste(imageDirPath, chptImageB, sep = ""))
```

7. Error learning.
   Here we begin the first DADA2-specific processing step: error-learning. For earror-learning I set randomize to `TRUE`, otherwise it takes the samples in the order they are in the metadata list, until the number of bases are reached, meaning that error will never take into account those generated by specific amplicons that may come later in longer lists.
   Note: In the case where we have low numbers of merged read pairs compared to processed unmerged sequences, it is suggested to increase `maxMismatch` value during the `mergePairs` step (https://github.com/benjjneb/dada2/issues/648).

- errorLearningPool1. Collect the set of filtered fastq files into a list:

```
filtFs <- metadatafiltPlate1$filtFwd
filtRs <- metadatafiltPlate1$filtRev
```

- Extract the names of the fastq files, and compare to ensure there are files for both forward and reverse reads. Once this is confirmed, assign the forward and reverse fastq file pairs the same sample name, using the forward reads as sample name template:

```
sampleNamesF <- sapply(strsplit(basename(filtFs), "_F_filt.fastq.gz"), `[`, 1)
sampleNamesR <- sapply(strsplit(basename(filtRs), "_R_filt.fastq.gz"), `[`, 1)
if(!identical(sampleNamesF, sampleNamesR))
  stop ("Forward and reverse files do not match.")
sampleNames   <- sampleNamesF
names(filtFs) <- sampleNames
names(filtRs) <- sampleNames
```

- Set Rs random number generator:

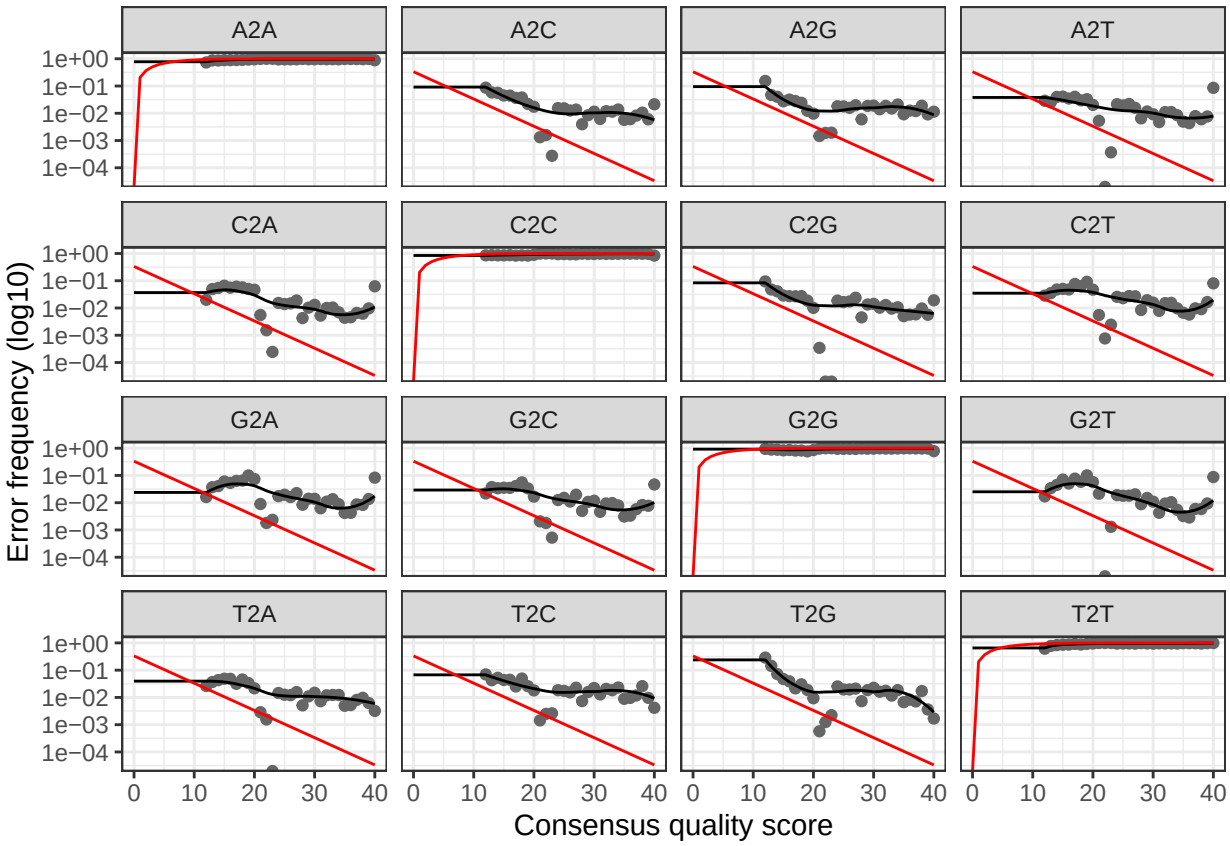
```
set.seed(100)
```

- Learn forward and reverse error rates:

```
library("dada2")
# Learn forward error rates:
errF <- dada2::learnErrors(filtFs, nbases = 1e8, randomize = TRUE,
                           multithread = TRUE, verbose = TRUE)
# Learn reverse error rates:
errR <- dada2::learnErrors(filtRs, nbases = 1e8, randomize = TRUE,
                           multithread = TRUE, verbose = TRUE)
```

We can visualise the estimated error rates as a sanity check:

```
dada2::plotErrors(errF, nominalQ = TRUE)
```



- drepDadaMergePool1. Sample inference and merging of paired-end reads:

```
mergers <- vector("list", length(sampleNames))
names(mergers) <- sampleNames
ddFs <- vector("list", length(sampleNames))
names(ddFs) <- sampleNames
ddRs <- vector("list", length(sampleNames))
names(ddRs) <- sampleNames
```

```
for(sam in sampleNames) {
  cat("Processing:", sam, "\n")
    derepF <- dada2::derepFastq(filtFs[[sam]])
    ddF    <- dada2::dada(derepF, err = errF, multithread = TRUE, verbose = TRUE)
    ddFs[[sam]] <- ddF
    derepR <- dada2::derepFastq(filtRs[[sam]])
    ddR    <- dada2::dada(derepR, err = errR, multithread = TRUE, verbose = TRUE)
    ddRs[[sam]] <- ddR
    merger <- dada2::mergePairs(ddF, derepF, ddR, derepR,
                          maxMismatch = 1, verbose = TRUE)
    mergers[[sam]] <- merger
}
rm(derepF); rm(derepR)
save.image(paste(imageDirPath, chptImageB, sep = ""))
```

We need to remove those samples that have no remaining reads:

```
dfToKeep    <- mergers[sapply(mergers, function(x) any(x$abundance > 0))]
mergersKept <- dfToKeep
samNamesKeptMergers <- names(mergersKept)

# See which samples were dropped because they had no remaining reads:
names(mergers[sapply(mergers, function(x) all(x$abundance == 0))])
```

```
[1] "ITS_NA_GB_S_5_FD_B" "ITS_NA_GB_S_6_FD_A"
```

Construct sequence table and print it to file:

```
seqTabKeptMergers <- dada2::makeSequenceTable(mergersKept)
saveRDS(seqTabKeptMergers, paste(sharedPathReg, region,
                                 "_", metadataRegion$SeqPlate[1],
                                 "_seqtab_KeptSamples.rds", sep = ""))
save.image(paste(imageDirPath, chptImageB, sep = ""))
```

9. mergeSplitRuns. **Note:**
   This next step is not needed for our ITS set since they were all sequenced on the same run.
10. remChimeric. Remove chimeric sequences from the sequence table.

```
stAllKept <- readRDS(paste(sharedPathReg, region,
                                 "_seqtab_KeptSamples.rds", sep = ""))
seqTabKept <- dada2::removeBimeraDenovo(stAllKept, method = "consensus",
                                        multithread = TRUE, verbose = TRUE)
save.image(paste(imageDirPath, chptImageB, sep = ""))
# Identified 4974 bimeras out of 15010 input sequences.
```

Let's see how many sequences we lost after removing chimeric sequences:

```
[1] 0.07301167
```

```
# [1] 0.07301167
```

Good - we lost less than 8% when removing chimeric sequences. Also - notice that this number doesn't change if we compare the set with and without the samples with no reads removed, which is what we expect.

Note that the samples we removed from the mergers dataframe list are not included in the summary table since they had no remaining reads by the end of read processing.
Overview of counts througout:
For this set, we will use the mergers without empty samples removed, so that we can see the reads remaining throughout our pipeline.

```
# Set a simple function:
GetN <- function(x) sum(getUniques(x))
# Keep only sample names remaining in seqTabKept
seqTabKeptNames <- row.names(seqTabKept)
sampleNamesKept <- intersect(sampleNames, seqTabKeptNames)
# Keep only rows in trimOut that match value in seqTabKeptNames vector:
trimOutKept <- subset(trimOut, rownames(trimOut) %in% seqTabKeptNames)
# Keep only elements in ddFs and ddRs that are in the seqTabKeptNames vector:
ddFsKept <- ddFs[seqTabKeptNames]
ddRsKept <- ddRs[seqTabKeptNames]
# Generate a simple table for tracking:
summaryTbl <- data.frame(row.names = sampleNamesKept,
                         input = trimOutKept[,1],
                         filtered = trimOutKept[,2],
                         dadaF = sapply(ddFsKept, GetN),
                         dadaR = sapply(ddRsKept, GetN),
                         merged = sapply(mergersKept, GetN),
                         nonchim = rowSums(seqTabKept),
                         finalPercReadsKept = round(
                           rowSums(seqTabKept)/trimOutKept[,1]*100, 1))
rownames(summaryTbl) <- sampleNamesKept
head(summaryTbl)
```

```
                   input filtered dadaF dadaR merged nonchim
ITS_NA_GB_B_1A_PS   83207    51940 48337 47792  29146   27827
ITS_NA_GB_B_1A_PW   94513    54133 51500 51017  35467   34068
ITS_NA_GB_B_1B_NS    9658     7828  6452  6659   4853    4831
ITS_NA_GB_B_1B_PS   15334    10003  8956  8503   4570    4238
ITS_NA_GB_B_1B_PW   44202    30674 27367 26991  15603   15334
ITS_NA_GB_B_2A_NS_B 62190    39752 37136 37238  26784   25087
                   finalPercReadsKept
ITS_NA_GB_B_1A_PS                33.4
ITS_NA_GB_B_1A_PW                36.0
```

```
ITS_NA_GB_B_1B_NS                50.0
ITS_NA_GB_B_1B_PS                27.6
ITS_NA_GB_B_1B_PW                34.7
ITS_NA_GB_B_2A_NS_B              40.3
```

Write our tracking results into a csv file:

```
write.csv(summaryTbl,
          file = paste(sharedPathReg, "readCountTracking.csv", sep = ""),
          row.names =TRUE)

# Show the samples that have no remaining reads by the end of processing:
subset(sampleNames, !(sampleNames %in% sampleNamesKept))
```

```
[1] "ITS_NA_GB_S_5_FD_B" "ITS_NA_GB_S_6_FD_A"
```

Generate a final metadata table that includes only those samples that had reads surviving the processing steps:

```
library("data.table")
data.table::setkey(metadatafilt, "LibraryName")
samples <- sampleNamesKept
metadataFinal <- metadatafilt[samples]

# Write our final metadata table to a csv file:
sampleTblName <- "final_processed_metadata"
write.table(metadataFinal,
            paste(sharedPathReg, sampleTblName, ".csv", sep = ""),
            sep = ",", row.names = FALSE, quote= FALSE)
```

11. assignTax. Assign taxonomy - UNITE database for ITS sequences:

```
taxTab <- dada2::assignTaxonomy(seqTabKept, refFasta = uniteSetSh,
                                multithread = TRUE, verbose = TRUE)
# Finished processing reference fasta. 874 out of 34286 were assigned to the species level.
# Of which 788 had genera consistent with the input table.
```

12. Inspect the taxonomic assignments:

```
taxaPrint <- taxTab  # Removing sequence rownames for display only
rownames(taxaPrint) <- NULL
head(taxaPrint)
```

```
     Kingdom    Phylum          Class                Order
[1,] "k__Fungi" NA              NA                   NA
[2,] "k__Fungi" "p__Ascomycota" "c__Leotiomycetes"   "o__Helotiales"
[3,] "k__Fungi" "p__Ascomycota" "c__Sordariomycetes" "o__Hypocreales"
[4,] "k__Fungi" "p__Ascomycota" "c__Dothideomycetes" NA
[5,] "k__Fungi" "p__Ascomycota" "c__Leotiomycetes"   "o__Helotiales"
[6,] "k__Fungi" "p__Ascomycota" "c__Leotiomycetes"   "o__Helotiales"
     Family            Genus             Species
[1,] NA                NA                NA
[2,] "f__Leotiaceae"   "g__Alatospora"   "s__acuminata"
[3,] "f__Hypocreaceae" "g__Trichoderma"  "s__harzianum"
[4,] NA                NA                NA
[5,] "f__Helotiaceae"  "g__Filosporella" "s__annelidica"
[6,] "f__Helotiaceae"  "g__Filosporella" "s__annelidica"
```

Extract the standard goods from R:

```
# Let's give our sequence headers more manageable names (ASV_1, ASV_2,...)
asvSeqs <- colnames(seqTabKept)
asvHeaders <- vector(dim(seqTabKept)[2], mode = "character")
for (i in 1:dim(seqTabKept)[2]){
  asvHeaders[i] <- paste(">ASV", i, sep = "_")
}
# Making and writing out a fasta of our final ASV sequences:
  asvFasta <- c(rbind(asvHeaders, asvSeqs))
write(asvFasta, paste(sharedPathReg, "ASVs.fa", sep = ""))
# Write out our count table:
asvTab <- t(seqTabKept)
row.names(asvTab) <- sub(">", "", asvHeaders)
write.table(asvTab, paste(sharedPathReg, "ASVs_counts.txt", sep = ""),
            sep = "\t", quote = FALSE, col.names = NA)
# Write out our tax table:
asvTax <- taxTab
row.names(asvTax) <- sub(">", "", asvHeaders)
write.table(asvTax, paste(sharedPathReg, "ASVs_taxonomy.txt", sep = ""),
            sep = "\t", quote = FALSE, col.names = NA)
```

13. Construct phylogenetic tree
    Phylogenetic relatedness is commonly used to inform downstream analyses, especially the calculation of phylogeny-aware distances between microbial communities. The DADA2 sequence inference method is reference-free, so we must construct the phylogenetic tree relating the inferred sequence variants de novo. We begin by performing a multiple-alignment using the DECIPHER R package (Wright 2015).

```
library("DECIPHER")
seqs        <- dada2::getSequences(seqTabKept)
names(seqs) <- seqs # This propagates to the tip labels of the tree
```

```
algn <- DECIPHER::AlignSeqs(Biostrings::DNAStringSet(seqs),
                            anchor = NA, verbose = TRUE)
```

The phangorn R package is then used to construct a phylogenetic tree. Here we first construct a neighbor-joining tree, and then fit a GTR+G+I (Generalized time-reversible with Gamma rate variation) maximum likelihood tree using the neighbor-joining tree as a starting point.

```
library("phangorn")
phangAlign <- phangorn::phyDat(as(algn, "matrix"), type = "DNA")
phangorn::write.phyDat(phangAlign,
                       file = paste(sharedPathReg, "alignedSeqs.fasta", sep = ""),
                       format = "fasta")
dm     <- phangorn::dist.ml(phangAlign)
treeNJ <- phangorn::NJ(dm) # Note, tip order != sequence order
fit    <- phangorn::pml(treeNJ, data = phangAlign)
fitGTR <- update(fit, k= 4, inv = 0.2)
ape::write.tree(fitGTR$tree, file = paste(sharedPathReg, "pre_GTR.phy", sep = ""))

fitGTR <- phangorn::optim.pml(fitGTR, model = "GTR", optInv = TRUE,
                              optGamma = TRUE, rearrangement = "NNI",
                              control = pml.control(trace = 0))
ape::write.tree(fitGTR$tree, file = paste(sharedPathReg, "GTR.phy", sep = ""))
detach("package:phangorn", unload=TRUE)
```

Use this chunk to run the above 2 chunks as qsubs on the biocluster instead of interactively.

```
prefix <- paste("B_fitGTRTree_R_mergedSeqPlatesData", region, sep = "_")
cmd <- paste("load('", paste(imageDirPath, chptImageB, sep = ""), "')\n",
             'library("phangorn")\n',
             'seqs       <- dada2::getSequences(seqTabKept)\n',
             'names(seqs) <- seqs\n',
             'algn     <- DECIPHER::AlignSeqs(Biostrings::DNAStringSet(seqs),
                                              anchor = NA, verbose = TRUE)\n',
             "save.image('", paste(imageDirPath,
                                   "fitGTR_", chptImageB, sep = ""), "')\n",
             'phangAlign <- phangorn::phyDat(as(algn, "matrix"), type = "DNA")\n',
             'phangorn::write.phyDat(phangAlign,
                                     file = paste(sharedPathReg,
                                     "alignedSeqs.fasta", sep = ""),
                                     format = "fasta")\n',
             'dm     <- phangorn::dist.ml(phangAlign)\n',
             'treeNJ <- phangorn::NJ(dm)\n',
             'fit    <- phangorn::pml(treeNJ, data = phangAlign)\n',
             'fitGTR <- update(fit, k = 4, inv = 0.2)\n',
             "ape::write.tree(fitGTR$tree, file = '",
                        paste(sharedPathReg, "pre_GTR.phy",
                              sep = ""), "')\n",
```

```
            "save.image('", paste(imageDirPath, "fitGTR_",
                                  chptImageB, sep = ""), "')\n",
            'fitGTR <- phangorn::optim.pml(fitGTR, model = "GTR",
                                           optInv = TRUE, optGamma = TRUE,
                                           rearrangement = "NNI",
                                           control = pml.control(trace = 0))\n',
            "ape::write.tree(fitGTR$tree, file = '",
                             paste(sharedPathReg, "GTR.phy", sep = ""), "')\n",
            "save.image('", paste(imageDirPath,
                                  "fitGTR_", chptImageB, sep = ""), "')\n",
            'detach("package:phangorn", unload = TRUE)',
            sep = "")
MakeRQsubs(cmd, prefix)
```

If the above chunk was used to align and get the tree, update the environment once it is done so that the final fitGTR can be updated. To do this, simply load the image that was written in the qsub script in the above chunk.

```
load(paste(imageDirPath, "fitGTR_", chptImageB, sep = ""))
```

14. Combine data into a phyloseq object
    Collect the data from the sequence table and the metadata table such that they can be linked by a common column called "SampleID", with only those columns we'll require. Then write out this table:

```
sampleDatadf <- as.data.frame(metadataFinal)
sampleDatadf$SampleID <- sampleDatadf$LibraryName

all(rownames(seqTabKept) %in% sampleDatadf$SampleID)

rownames(sampleDatadf) <- sampleDatadf$SampleID
keepCols <- c("Region", "Sample", "Provider", "Experiment",
              "ExtractionKit", "LibraryName", "SampleID")

sampleDataDF <- sampleDatadf[rownames(seqTabKept), keepCols]

# write the sample data dataframe to csv:
write.table(sampleDataDF,
            paste(sharedPathReg, sampleTblName, "_forPhyloseq.txt", sep = ""),
            sep = "\t", quote = FALSE, col.names = NA)
```

Prepare the phyloseq objects - one that includes the data obtained from the distance tree, and one that doesn't (in case the distance tree is not yet created):

```
# phyloseq object that does not include tree:
phySeq   <- phyloseq::phyloseq(otu_table(seqTabKept, taxa_are_rows = FALSE),
                               sample_data(sampleDataDF),
                               tax_table(taxTab))

# phyloseq object that DOES include tree:
ps <- phyloseq::phyloseq(otu_table(seqTabKept, taxa_are_rows = FALSE),
                         sample_data(sampleDataDF),
                         tax_table(taxTab), phy_tree(fitGTR$tree))
```

The only variable we need to save from the fitGTR image is "ps". Save this individual variable, and then load it with the chptImageB, without all the intermediate varibales.

```
# Save the ps variable:
save(ps, file = paste(imageDirPath, "ps.Rdata", sep = ""))

# Clear the global environment, including hidden variables:
rm(list = ls(all.names = TRUE))

# Load the chptImageB and the ps variable:
load("/isilon/cfia-ottawa-fallowfield/users/girouxeml/GitHub_Repos/r_environments/ecobiomics/e
load(paste(imageDirPath, "ps.Rdata", sep = ""))

# Save the chptImageB again so it includes the ps variable:
save.image(paste(imageDirPath, chptImageB, sep = ""))
```