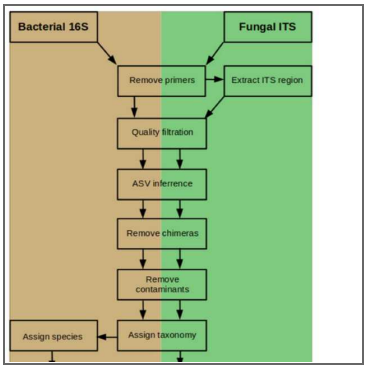

NOV 10, 2023

OPEN ACCESS

DOI:
dx.doi.org/10.17504/protocols.io.q26g7pw63gwz/v1

Protocol Citation: Geoff Zahn 2023. ITS amplicon computational workflow . **protocols.io** https://dx.doi.org/10.17504/protocols.io.q26g7pw63gwz/v1

MANUSCRIPT CITATION: Zahn, G. (2022). Marker Genes (16S and ITS) Protocol for Plant Microbiome Analyses. *BIO-PROTOCOL*, *12*(8). https://doi.org/10.21769/BioProtoc.4395

ITS amplicon computational workflow

Geoff Zahn[1]

[1]Utah Valley University

Marine Fungi | Zahn

Geoff Zahn

ABSTRACT

Many research questions in plant science depend on surveys of the plant microbiome. When the questions depend on answering "who is there" rather than "what are they doing," the most cost-effective method for interrogating microbiomes is often via targeted meta-amplicon surveys. There are numerous platforms for processing and analyzing meta-amplicon data sets, but here we will look at a flexible, reproducible, and fully customizable pipeline in the R environment. This approach has the benefit of being able to process and analyze your data in the same setting, without moving back and forth between standalone platforms, and further benefits from being completely flexible in terms of analyses and visualizations you can produce, without being limited to pre-selected tools available in point-and-click analysis engines, such as QIIME, Galaxy, or MG-RAST.

ATTACHMENTS

Zahn - 2022 - Marker Genes (16S and ITS) Protocol for Plant Micr.pdf

GUIDELINES

All scripts for this computational "recipe" can be found at: https://github.com/Bio-protocol/metaamplicon-recipe

Protocol status: Working
We use this protocol and it's working

Created: Nov 10, 2023

Last Modified: Nov 10, 2023

PROTOCOL integer ID: 90779

MATERIALS

Software

1. cutadapt (Martin, M., 2011; version 2.10; https://cutadapt.readthedocs.io/en/stable/changes.html#v2-1-2019-03-15)
2. ITSxpress (Rivers *et al.*, 2018; version 1.0; https://github.com/usda-ars-gbru/itsxpress)
3. R (R Core Team, 2017; version 3.6.3; https://www.R-project.org/)
4. tidyverse R package (Wickham *et al.*, 2019; version 1.3.0; https://www.tidyverse.org/)
5. DADA2 R package (Callahan *et al.*, 2016; version 1.14.1; https://benjjneb.github.io/dada2/)
6. decontam R package (Davis *et al.*, 2018; version 1.6.0; https://benjjneb.github.io/decontam/)
7. phyloseq R package (McMurdie and Holmes, 2013; version 1.30.0; https://joey711.github.io/phyloseq/)
8. Biostrings R package (Pagès *et al.*, 2021; version 2.54.0; http://www.bioconductor.org/packages/release/bioc/html/Biostrings.html)
9. phangorn R package (Schliep, 2011; version 2.2.5; https://CRAN.R-project.org/package=phangorn)
10. msa R package (Bodenhofer *et al.*, 2015; version 1.18.0; https://bioconductor.org/packages/release/bioc/html/msa.html)
11. ShortRead R package (Morgan *et al.*, 2009; version 1.44.3; https://bioconductor.org/packages/release/bioc/html/ShortRead.html)
12. corncob R package (Martin, B. D. *et al.*, 2020; version 0.2.0; https://CRAN.R-project.org/package=corncob)
13. vegan R package (Okansen *et al.*, 2016; version 2.6.0; https://CRAN.R-project.org/package=vegan)
14. patchwork R package (Pedersen, 2020; version 1.0.1; https://CRAN.R-project.org/package=patchwork)

Input data

1. Input data are the demultimlexed fastq reads from sequenced bacterial 16S amplicons. There should be two files for each experimental sample that was sequenced; a file for forward reads, and a file for reverse reads. This workflow is written for the commonly used Illumina sequencing platform, but IonTorrent sequences can be used with some slight modification (noted in the workflow).
2. Example data consists of custom subsets of host-associated bacterial 16S amplicons taken from 46 seagrass samples (including PCR negatives). Files have been truncated and modified from their original form to allow for faster computational times, while still showing significant differences in community composition between sampling groups. They no longer represent any real study system, but are meant to be used as a teaching tool. Raw and intermediate data are provided in the GitHub repository associated with this publication.

BEFORE START INSTRUCTIONS

All scripts for this computational "recipe" can be found at: https://github.com/Bio-protocol/metaamplicon-recipe

Remove primers

1 The first step is to remove any PCR primers that may still be present on your reads. Often, sequencing centers remove these by default, but it's a good idea to check. The R script "00_Remove_Primers.R" walks you through this process. In short, you provide the PCR primers
that we used for generating our amplicons and then use the *cutadapt* tool to search for their presence in all possible orientations. They are then removed and we can proceed with quality filtration.
This process first removes all reads with ambiguous (N) bases, and then removes primer sequences and adaptors from remaining reads. The newly modified reads are stored in a separate directory (in this case ./data/filtN/). When working with bacterial 16S amplicons, this step can generally be accomplished by simply trimming the first N bases from both forward and reverse reads. However, using *cutadapt* is a more generalized approach that also accounts for mis-oriented reads, does not require you to know ahead of time if your primers are present or not, and is thus recommended.

Extract ITS region

2 It's important that we extract the ITS region of interest at this point. The problem is that while ITS1 or ITS2 are good at predicting fungal taxonomy, the primers used to amplify these regions sit inside highly conserved flanking regions (Gardes and Bruns, 1993). Since ITS lengths can vary so greatly (Bengtsson-Palme et al., 2013), keeping these conserved regions in our sequences can greatly bias downstream taxonomic assignment algorithms. A solution exists though, in the *itsxpress* software (Rivers et al., 2018), which can identify and trim these conserved regions from the ends of our ITS reads. This is typically run via the command line, but we can do it from within R. The supplementary file "00_Trim_ITS_Region.R" gives an thorough walkthrough example of how to extract the ITS1 region.

Quality filtration

3 Once you are sure your primers aren't present in your reads (or flanking conserved regions if analyzing fungal ITS data), it's time to do some quality control work to filter out ambiguous and low-quality reads, and to trim reads where quality begins to suffer. The R script "01_Process_Raw_16S_Reads.R" goes through this step by step, but it's worth our time to explain what's going on. The workhorse software package for this and subsequent steps is DADA2 (Callahan et al., 2016).

We first parse our files generated by cutadapt (and itsxpress, in the case of fungal amplicons). The only filtration done on these so far is to remove ambiguous reads and any remaining primer sequences. The file parsing needs to account for separate forward and reverse reads, and it does so by matching patterns in the file names. In our example data forward reads are denoted by "…pass_1" and reverse

reads are denoted by "…pass_2" in their names. This can vary between sequencing centers, and your reads may have different patterns such as "R1" and "R2" or "F" and "R." Be sure to modify the example code to match your forward and reverse reads accordingly where noted.

The plotQualityProfile() function from the DADA2 package takes a look inside these sequence files and allows you to make an informed judgement about how to filter and trim your raw reads. For example, in Fig. 2, we are looking at the separate quality profiles of the forward and reverse reads of two samples. The X-axis shows read length, and the Y-axis shows summary information of quality scores for every read in the file. The green line is the mean, the orange line is the median, and the dashed orange lines are the 25th and 75th quantiles. It's up to you to determine where good cutoff points might be for the forward and reverse reads. Keep in mind that in order to merge the reads later on, they will need to have sufficient overlap (typically >50 bases).

If you are working with fungal ITS reads, it is typical that reverse reads aren't of sufficient quality to merge with forward reads. Especially with variable ITS lengths, it is somewhat common practice to forego using reverse reads, and to just process the forward reads output by *itsxpress* (Darcy *et al.*, 2020; Tipton *et al.*, 2019, 2021).

The trimming and filtration step will discard some reads; how many depends on how stringently you set the parameters. If you notice that the majority of your reads are being discarded, you might try relaxing the *maxEE* parameter a bit.

ASV inferrence

4 Now that we have a set of quality-controlled sequence files, we're ready to perform a "denoising" step that can predict and correct likely sequencing errors. This is an important step when analyzing exact "amplicon sequence variants" (ASVs) as opposed to "operational taxonomic units" (OTUs). There is always a bit of "noise" in sequence data, representing sequence errors. OTUs were, in part, conceived of as a way to deal with this constant threat of sequencing noise. By clustering highly similar sequences together, it accounts for small amounts of variation. However, much of that variation is real and informative (Callahan et al., 2017; Tipton et al., 2021). Methods now exist to discover and correct sequence errors, allowing us to use ASVs. The advantages include finer taxonomic resolution, the potential to discover patterns that might be hidden by artificially clustering sequences together, and most importantly, the ability to easily combine data from multiple studies. Since each ASV is an actual sequence, rather than an undefined cloud of sequences (as with OTUs), they are directly comparable outside of a given study. This facilitates reproducibility and re-use such as in meta-analyses.

The DADA2 R package gives us a set of tools for inferring these ASVs. First, we build a mathematical profile of error rates along the length of our reads. In basic terms, we can think of this error profile as being a mathematical representation of our quality profile plots (Figure 3). The learnErrors() function is a machine-learning tool that builds an error model based on your data. Figure 4 shows this error model.

What you're looking for is a decent fit between observed and estimated errors, and a decline in error as quality scores increase; It doesn't need to be a perfect fit. If your model looks reasonable (as ours does in Figure 4), you are ready to use that error model to correct sequence errors. A poorly fit error model (indicated by large discrepancies between observed points and the model fit line) may be a result of binned quality scores or a lack of 'diversity' of quality scores. In this case, be sure you are using the raw sequence read files for creating the error profile, not reads that have been pre-processed in any way.

The dada() function performs denoising based on the estimated error model. In essence, it is performing statistical tests that determine whether a given sequence was observed too many times to have been caused by sequencing errors in currently inferred sequences (Rosen et al., 2012). The dada() function is run separately on forward and reverse reads and the output is a set of sequence variants that were inferred from the "noisy" input sequences. From this point, we are ready to merge our forward and reverse reads and construct a sequence table. These tasks are accomplished with the mergePairs() and makeSequenceTable() functions, respectively.

5

Remove chimeras

Chimeric sequences result from incomplete extension during PCR. Partial amplicons can serve as "false templates" for primers to bind to, resulting in reads that are composed of more than one true sequence. Chimera detection involves evaluating distinct matches between the left-side and right-side of reads and potetial "parent" source reads. In practice, this detection and removal step can be conducted with the *removeBimeraDenovo()* function as shown in the R script 01_Process_Raw_16S_Reads.R."

6

Remove contaminant sequences

While sadly not universally practiced, including and sequencing negative controls is a crucial step when performing meta-amplicon studies. Often, these control samples are blank DNA extractions that are then sequenced. This allows us to discover DNA sequences that were present in the lab environment that do not necessarily represent meaningful diversity in our real samples. The decontam package (Davis et al., 2018) has been implemented to deal with these negative controls, using them to detect likely contaminant sequences. The way we will use this package is by giving the isContaminant() function our newly created sequence table and a list of which samples are negative controls. Your sample metadata spreadsheet likely contains study-relevant information about each sequenced sample, but should also include a column denoting whether a sample is a negative control (Table 2 shows an example metadata sheet formatted correctly). Sequences with increased prevalence in control samples can then be identified and removed. Once potential contaminants are removed, we can then remove the negative control samples from further downstream steps since their job is done.

Assign taxonomy

7 Assigning taxonomy to DNA barcodes is a complicated subject. The success and accuracy of any assignments depend on the algorithms and databases that you use. In fact, with phylogeny-independent taxonomic assignments, the current limiting factors are the completeness and accuracy of the databases to which we match our sequences. With this in mind, it is critical that you carefully consider what methods to use. The RDP Naive Bayesian Classifier algorithm (Wang et al., 2007) is one such algorithm that is widely used and has several advantages. First, it does not rely on sequence alignments, which makes it suitable for fungal ITS studies. Second, it is incredibly fast compared to algorithms such as BLAST. Third, it provides bootstrapped confidence values to all assignments and allows matching to the "least common ancestor," reducing the incidence of false positive matches. Its performance for species-level taxonomic assignments of bacteria is lacking, and new information has shown that exact 100% matching is the only justifiable method for assigning species to bacterial ASVs (Edgar, 2018). A selection of properly formatted reference files for popular taxonomic databases are available at http://benjjneb.github.io/dada2/training.html. Here, we use two complementary methods to assign taxonomy to our reads: 1) the RDP Classifier using the RDP 16S Training Set database, implemented via the assignTaxonomy() function, and 2) exact 16S matching for species-level assignments implemented via the addSpecies() function. The "Assign Taxonomy" section of "01_Process_Raw_16S_Reads.R" script covers these steps. These functions take our sequence table as input and return a data frame matching our ASVs to taxonomy at the deepest levels that could be assigned unambiguously.

If you are using fungal ITS reads, taxonomic assignment is essentially the same, but you will need to use a eukaryotic database such as the UNITE Eukaryotic Database (DOI: 10.15156/BIO/2938069) instead of a bacterial 16S database, and the addSpecies() function should be omitted.

The last step in this script is to construct a phyloseq object in which to store your ASV table along with your sample metadata and taxonomic assignments. The phyloseq package (McMurdie and Holmes, 2013) provides a data structure that can combine a sequence abundance table (like our ASV table), taxonomic information, sample metadata, and a phylogenetic tree into one object that makes working with these disparate tables much simpler.