Bioinformatics v3.0

Suggested bioinformatics for the processing of sequence data from libraries containing 16S and ITS amplicons, coligos, and an internal standard. This suggested pipeline follows on a lot of work conducted by many members of our group.

Demultiplexing will be performed by GTL staff and will not be discussed in detail below. Current demultiplexing code uses Levenshtein distances to correct up to two errors in MIDs. For further details, contact Josh.

Note to readers: I often use variables such as 'YOURFILE' (or similarly generic sounding file names) to hold the space for input and output in the commands below. These variables will need to be switched out with your own file names. Also, I do not wrap many of the commands in loops, so that they can be applied to all files in a directory. If you do not know how to do that, please look up how to use 'for' loops in bash and contact me for further help (also there are a few examples of for loops below). I am happy to help if I can.

@ Abby Hoffman @ Macy Ricketts @ Seifeddine Ben Tekaya @ Alex Buerkle @ Gordon Custer @ Ella DeWolf @ Reilly Dibner @ Erin Bentley If I have missed someone actively working on data, please tag them. Thanks!

Snapshot of all code in this document with only minimal statements regarding the purpose of the code (@ Alex Buerkle). For detailed comments one each step, see the portion of this page following this snapshot.

The output of each step is the input of the next step, with a few exceptions that should be clear from the notes below.

I do not specify an integer for cores in any of these commands. The teton-cascade nodes have 40 cores versus the regular teton nodes having 32. So modest gains could be had from using the cascade nodes. Benchmarking by ARCC suggests that the teton-knl nodes, which have 72 cores, actually don't offer a speed-up because each core is multi-threaded and this seems to slow things down. Also, vsearch doesn't seem to offer multi-node support so we can't go crazy and use 1024 cores (which is the current supported max).

Step 1. load modules

```
module load cutadapt/2.10 usearch/10.0.240 vsearch/2.15.1
```

The only other software that will be used is fastp, which is located here: `/project/microbiome/bin/fastp`

Step 2. Remove primers using cutadapt and clean up headers

Remove forward 16s primer

```
cutadapt -g ^GTGYCAGCMGCCGCGGTAA -o FWD_OUTPUT FWD_INPUT -e 0.25 --cores=N
```

Remove reverse 16s primer

```
cutadapt -g ^GGACTACHVGGGTWTCTAAT -o REV_OUTPUT REV_INPUT  -e 0.25 --cores=N
```

Remove forward ITS primer

```
cutadapt -g ^CTTGGTCATTTAGAGGAAGTAAT -o REV_OUTPUT REV_INPUT -e 0.25 --cores=N
```

Remove reverse ITS primer

```
cutadapt -g ^GCTGCGTTCTTCATCGATGC -o REV_OUTPUT REV_INPUT  -e 0.25 --cores=N
```

Clean up the headers of all files to remove problematic characters.

```
sed 's/\s/_/g' INPUT | sed -e 's/^@16/@rna16/' - | sed -e 's/-/_/g' > OUTPUT
```

Step 3. Make coligo table

Note, I changed my mind and think it makes sense to include coligo processing in any comprehensive bash script, so long as we continue to use coligos.

The input to this call to awk is the output from step 2 (primers removed and headers cleaned up with sed).

```
awk '{if (/^@[[:alnum:]_]+$/) {print}else{print substr ($0, 0, 13)}}' INPUT > OUTPUT
```

```
vsearch --fastq_filter INPUT --fastaout OUTPUT
```

Output of the following command is our coligo table. It is left to the client to look at the coligo table, we don't process it further here.

```
vsearch -search_exact INPUT -db /project/microbiome/ref_db/coligos_and_abbreviatedISD.fa -strand plus -
otutabout OUTPUT -minseqlen 5 -threads N
```

Here is all of it as one line:

```
awk '{if ($1 ~ /@/) {print}else{print substr ($0, 0, 13)}}' INPUT |
vsearch --fastq_filter - --fastaout - | vsearch --search_exact - --db /project/microbiome/ref_db
/coligos_and_abbreviatedISD.fa --strand plus --otutabout OUTPUT --minseqlen 5 --threads 32
```

Step 4. Remove low complexity reads (this gets rid of coligos)

```
usearch -filter_lowc FWD_INPUT -reverse REV_INPUT -output FWD_OUTPUT -output2 REV_OUTPUT
```

There doesn't appear to be a similar command in vsearch.

Step 5. Merge reads (note we are saving both merged and unmerged reads so there are multiple outputs to this command)

```
vsearch --fastq_mergepairs FWD_INPUT --reverse REV_INPUT --fastqout FWD_OUTPUT --fastq_maxdiffs 12 --
fastq_allowmergestagger --fastq_minovlen 10 --fastq_minmergelen 60 --fastqout_notmerged_fwd
FWD_OUTPUTnotmerged --fastqout_notmerged_rev REV_OUTPUTnotmerged --threads N
```

Step 6. Filter reads

Filter merged reads using:

```
vsearch -fastq_filter INPUT -threads N -fastq_maxee 1 -fastaout OUTPUT
```

Filter reads that didn't merge using:

```
/project/microbiome/bin/fastp --in1 FWD_INPUT --in2 REV_INPUT -q 15 -u 40 -l 107 --out1 FWD_OUTPUT --out2
REV_OUTPUT --thread N
```

Step 7. Truncate and join reads that didn't merge

```
vsearch -fastx_filter FWD_INPUT --fastq_trunclen 215 -fastqout FWD_OUTPUT -threads N

vsearch -fastx_filter REV_INPUT --fastq_trunclen 215 -fastqout REV_OUTPUT -threads N

vsearch -fastq_join FWD_INPUT -reverse REV_INPUT -fastaout OUTPUT -threads N
```

Step 8. Dereplicate reads

I think it makes sense to cat both merged (output from step 5 above) and joined reads (output from step 7 above) and pipe them into this command. Otherwise the command must be run twice, once on the merged reads then again on the joined reads and then the output of both of those commands should be concatenated.

```
vsearch -derep_fulllength INPUT --output OUTPUT --sizeout --sizein
```

Step 9. Make OTUs and remove chimeras

```
vsearch --cluster_unoise INPUT --centroids OUTPUT --minsize 8 --relabel 'otu' --sizein --sizeout

vsearch --uchime3_denovo INPUT --nonchimeras OUTPUT
```

Step 10. Make OTU table

The input to this is concatenated merged and joined reads (reads should be filtered and in fasta format as output from steps 7 and 5 above).

The "-db" argument below stands for database and should be the fasta sequence for the OTUs/ESVs (from step 9) also note that the OUTPUT here is in single quotes.

```
vsearch --usearch_global INPUT --db OTUfastaESV --otutabout 'OUTPUTESV' --id 0.99 --threads N
```

Remove the pesky "#OTU ID" that vsearch puts in the first cell of the table:

```
sed -i 's/^#OTU ID/OTUID/' INPUT
```

Step 11. Generate taxonomic hypotheses

Input is OTU sequences

For ITS reads:

```
vsearch -sintax INPUT -db /project/microbiome/ref_db/unite4_02_20.fa -tabbedout OUTPUT -strand both -
sintax_cutoff 0.8 --threads N
```

For 16S reads

```
vsearch -sintax INPUT -db /project/microbiome/ref_db/gg_16s_13.5.fa -tabbedout OUTPUT -strand both -
sintax_cutoff 0.8 --threads N
```

Primer removal

We currently advocate primer removal. This may change at some point as there is some evidence to suggest some of the variation within primer sequences is biologically derived. For now, however, it is simplest to remove primers. This drastically cuts down the number of sequence variants within the data, which makes analysis much, much easier.

Cutadapt is an easy to use tool that searches for a specified sequence and can remove it from a string; it can handle IUPAC codes, which is nice, and uses a semi-global alignment algorithm. cutadapt is installed globally on Teton. Details for the software be found here: https://cutadapt.readthedocs.io/en/stable/algorithms.html. The '-e' flag that we use is *maximum error rate* and 0.25 corresponds to 25% mismatch. This seems reasonable to me, though I have not gone into the weeds testing different values here to see if results can be optimized. I am not convinced that such digging would be a good use of time right now.

For forward 16S:

```
cutadapt -g ^GTGYCAGCMGCCGCGGTAA -o OUTPUT INPUT (assuming a - for a piped input) -e 0.25 --cores=N
```

(note, sometimes this primer seems to be represented with only one A at the 3' end)

For reverse 16S:

```
cutadapt -g ^GGACTACHVGGGTWTCTAAT -o OUTPUT INPUT  -e 0.25 --cores=N
```

For forward ITS

```
cutadapt -g ^CTTGGTCATTTAGAGGAAGTAAT -o $OUTPUT INPUT -e 0.25 --cores=N
```

#Note, I added a trailing T here, because this variant occurs very often and if we do not include it many coligo sequences get messed up (since we take the bases following the primer to be coligo, if a bunch start with a T it messes up the search algorithm. The high error allowance we use here keeps this tweak from messing up our search for primers).

For reverse ITS:

```
cutadapt -g ^GCTGCGTTCTTCATCGATGC -o ${yclean}_primerRM INPUT  -e 0.25 --cores=N
```

It is convenient to save the sequences with primers removed because they are needed to generate an OTU table and to learn about cross-contamination. These are two parallel analyses that both need reads with no primers. After all analyses are complete then the reads with primers removed should be deleted.

The text in green represents a parallel analysis to learn about cross-contamination.

A note on processing coligos

Processing coligos is best done independently of the bioinformatics used to make an OTU table. This is because there are a fair number of coligo reads, so leaving them out during OTU table generation can save time. More importantly, a different alignment algorithm is needed to search sequences for coligos than is used for longer OTUs. This is because coligos are very short (only 13 nt long). Similarly, coligos don't merge very well. For all of these reasons, it is better to process coligos independently of the other sequences.

The goal of the coligo analysis is to make an OTU table that only has coligos in it. This makes for very easy analysis of cross-contamination. Here is a suggested way to make such a table.

#The following command will take some time to run. To test components of it, split it up and use just a few test reads (e.g., use head to pull a few reads from your data for testing).

```
cat *R1*primerremoved | sed -e 's/\s/_/g' - | sed -e 's/^>16/>rna16/' - | sed -e 's/-/_/g' >
allForwardReads.fastq

#remove the 13 bases. One could include this in the pipeline above if desired.

awk '{if (/^@[[:alnum:]_]+$/) {print}else{print substr($0, 0, 13)}}' allForwardReads.fastq > thirteenbases.
fastq


#Switch to vsearch functionality

module load vsearch
```

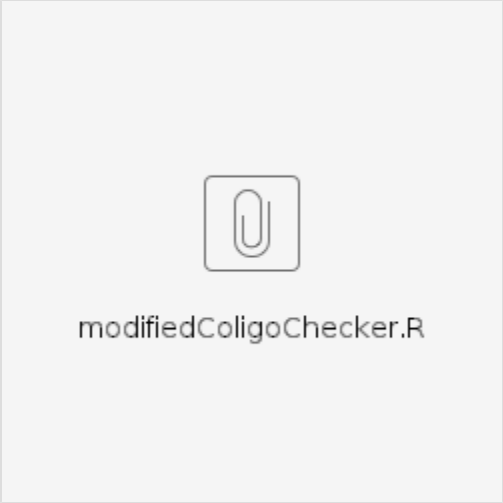
```
#convert to fasta

vsearch --fastq_filter thirteenbases.fastq -fastaout thirteenbases.fasta
```

#make coligo table using the search_exact algorithm. In a perfect world, we would probably use Levenshtein distances to error correct coligos and extract a bit more info. Given that the goal of this analysis is to detect very bad contamination, which should be apparent using this alignment algorithm, AND the vsearch command is easier to use and tweak then a custom perl/python script, I think it is best to use this simple approach for the sake of time and ease of use.

```
vsearch -search_exact thirteenbases.fasta -db /project/microbiome/ref_db/coligos_and_abbreviatedISD.fa -
strand plus -otutabout coligoTable -minseqlen 5 --threads N
```

The table output from vsearch will have a '#' starting the first line. Remove this and the space in “OTU ID” so that when you load the table into R the sample names will show up as the header (if desired).

I suggest searching through the table, using R or another language of your choice, to see when a coligo shows up somewhere it wasn't supposed to. Here is an R script that can be adapted to fit ones needs. The script is a draft and I made it for my own needs. It may be possible to codify this script so that can be used by all, but we will need to be very strict with sample names. If you have questions, please ask me and I can try to help.



As a final note, everyone will undoubtedly find instances of minor contamination are common. These are not something to worry about, as they will not affect most statistical analyses. If your analyses are qualitative then you will need to think more carefully about how to deal with cross-contamination. I am not convinced that qualitative analyses are the best choice for sequence data (please contact me if you are curious why), so proceed with caution.

Remove low-complexity sequences

This removes coligos so that we can save a bit of time during OTU table generation. It might also remove a few bogus sequences, but I doubt that provides much benefit.

```
usearch -filter_lowc FORWARD_withPrimerRemoved -reverse REVERSE_withPrimerRemoved -output
forwardLOWComplexityRemoved-output2 reverseLOWComplexityRemoved
```

I think this can be multi-threaded with ‘-threads’ but it is so fast that I didn't mess with that functionality.

Merge paired end reads

```
for f in *R1*polyGremoved; do fname=$(basename $f); fname2=${fname/R1/R2}; vsearch --fastq_mergepairs
${fname} --reverse ${fname2} --fastqout ${fname}.merged.fq --fastq_maxdiffs 12 --fastq_allowmergestagger --
fastq_minovlen 10 --fastq_minmergelen 60 --fastaout_notmerged_fwd ${fname}.NOTmergedFWD.fa --
fastaout_notmerged_rev ${fname}.NOTmergedREV.fa --threads N ; done
```

Note the cornucopia of flags here. I used

'fastq_maxdiffs” to set a maximum allowance of 12 mismatches. This number was chosen arbitrarily. Some might suggest this is high, however for reads that overlap a lot this might not be too bad, moreover vsearch tries to learn tricky base calls from data in both reads, thus I am ok having a high mismatch rate. Others may disagree, and if you do, please add a comment here explaining your results. It might be that we have fewer technical variants in our OTU list if we cut down the mismatchs allowed to just a few.

‘fastq_allowmergestagger’ lets paired reads extend past one another after alignment. The parts that ‘stick out’ are trimmed off. Sometimes we might expect stagger to occur, such as for our ISD and coligos, and sometimes for the locus of biological interest. Note that coligos should have been removed by the low complexity filtering.

'--fastq_minovlen' is the minimum overlap between paired-end reads. In our case, 10 bases at minimum have to overlap. I choose this parameter arbitrarily.

‘fastq_minmergelen’ is the minimum length of merged reads. I set this to 60 to cut out all the coligos during a previous version of this whole pipeline. Now coligos should be removed already, but I think it is best to cut short sequences anyway as they might be odd dimers or something.

'fastaout_notmerged” and similar - these output unmerged sequences.

‘threads’: allows for multithreading. When submitting a multi-threaded command via Slurm make sure you request enough cores (if you are not sure how to do this then check out this great page that ARCC made: https://arccwiki.atlassian.net/wiki/spaces/DOCUMENTAT/pages/377323550/Introduction+to+Job+Submission+01+Nodes+Tasks+and+Processors)

Note that I have moved away from using ‘fastq_truncqual’, which truncates sequences starting from the first base with the specified base quality score value or lower. This is because Novaseq only has 4 base calling probabilities and I figured that it might be difficult to avoid being too stringent and truncating sequences unnecessarily.

Important: we take the unmerged reads and concatenate them below. This is key for ITS, though probably not as important for 16S. The reason is that a lot of fungi have long ITS1 sequences that do not merge because the paired-end reads don’t overlap. For instance, the yeast in our mock community will not show up in the data if one relies on reads that merge only. Many mycologists just use forward reads for this reason, but this seems like an odd solution to me, given that reads can be concatenated and more information retained.

Filter reads that merged

I prefer to do this after merging, so that quality info from both reads can be used (dada2 attempts to correct errors before merging, which seems like a strange choice to me).

```
for f in *removed.merged*; do vsearch -fastq_filter ${f} -threads N -fastq_maxee 1 -fastaout ${f}.filtered.fa; done
```

‘fastq_maxee’ means at most one error is allowed. The probability of an error is calculated via base quality scores.

Filter reads that did not merge

Unfortunately v/usearch doesn’t seem to do a great job filtering single end reads. Consequently, I am using fastp here, because it seemed to handle filtering single-end reads better. I do think that usearch is better software, but this call to fastp seems to suffice.

```
/project/microbiome/bin/fastp --in1 FWD_INPUT --in2 REV_INPUT -q 15 -u 40 -l 107 --out1 FWD_OUT --out2 REV_OUT--thread N
```

- `-q, --qualified_quality_phred` the quality value that a base is qualified. Default 15 means phred quality >=Q15 is qualified.
- `-u, --unqualified_percent_limit` how many percents of bases are allowed to be unqualified (0~100). Default 40 means 40%
- -l is the minimum length required. I set this high to cut out ISD and coligos.
- --thread can be added for multi-threading.

In my experience a lot of reads fail filtering using this approach because reverse reads are not as good as forward reads and, since they aren’t merged, information cannot be shared between the members of a pair.

Truncate and join the reads that didn’t merge

Join is a synonym for concatenate, and is the terminology used by usearch. Before we join these reads we want to truncate them to a fixed length. This avoids the issue of making multiple ESVs for length variants. E.g., say we see a trailing ‘A' for one read that would otherwise be the same as some other read, we don’t want these both to be ESVs (at this point, I am assuming such minor length polymorphisms are not biological. I could be wrong, but missing the odd variant here and there won’t matter and I am going for simplicity in this pipeline…believe it or not ;). I picked 215 because it was a good round number and seemed to avoid weirdness that happens at the end of the read. If someone is keen it might be interesting to mess with this parameter some and see what happens.

usearch -fastx_truncate INPUTFORWARD -trunclen 215 -fastqout OUTPUT -threads N

usearch -fastx_truncate INPUTREVERSE -trunclen 215 -fastqout OUTPUT -threads N

usearch -fastq_join FWD_INPUT -reverse REV_INPUT -fastaout OUTPUT -threads N

I will refer to these as ‘joined’ reads henceforth. Note that vsearch has a 'fastq_join' function that does the same thing as the usearch command; using vsearch may be required if you have big files (because the free version of usearch is limited to 32 bits).

Dereplicate reads

Find unique sequences. Do this separately for 16S and ITS reads, since I imagine that you want to have a separate OTU table for both of these datasets.

```
cat YOURREAADS | vsearch -derep_fulllength - -threads 32 --output uniqueSequences.fa --sizeout --sizein
```

This output file is ordered by read counts, which is needed by the OTU algorithm.

Make OTUs

To make exact sequence variants use the UNOISE3 algorithm, like so:

usearch -unoise3 uniqueSequences.fa -zotus OUTPUT

#The default for this command is that sequences have to occur 8 times or more to be considered. Multi-threading is not available for this command at this point. Vsearch impliments this same algorithm using the command “--cluster_unoise”, if you use vsearch then the authors recommend using “--chime3_denovo” to remove chimeras afterwards. I am not sure if this is neccessary, but one could try it.

You will need to call variants for both 16s and ITS data, assuming that you would like to analyze those two loci separately.

NOTE: if you also want to make OTUs using some other clustering threshold, then consider using the ‘cluster_smallmem’ algorithm on denoised data (shown below). This seems to be reasonably fast and likely removes some spurious sequences from consideration. Data should be sorted by length so that OTUs that are a subset of larger OTUs can be combined. If you use this approach, then doublecheck that this combining of long and short OTUs worked as expected. I haven’t spent a lot of time on this yet as I prefer to work with exact sequence variants.

usearch -sortbylength esvs16s.fa -fastaout esvs16s_sorted.fasta -minseqlength 50

usearch -cluster_smallmem esvs16s_sorted.fa -id 0.97 -centroids otus16s_97id.fa --threads N

Chimeras

The usearch website has a lot about finding chimeras. Currently, the OTU creation algorithm looks for chimeras to some extent. However, Robert Edgar has done some simulation to show that finding chimeras reliably is extremely hard and potentially impossible. If you want to try removing more chimeras from your OTUs then consider the following command. At the moment, I am NOT using this, but I don’t think it would hurt to run it. Basically, I doubt that the decision to run this will affect inferences notably (of course, ESV richness will decline some if possible chimeras are removed, but richness should not be analyzed absolutely anyway, instead richness should only be compared among samples that were processed in the same way).

usearch -uchime_ref INPUT-db 16s_ref.udb (this needs to change for ITS, consider the UNITE databsae) -uchimeout OUTPUT -strand plus -mode sensitive

#the mode flag is interesting. If you go down this road then read the documentation to figure out what works best for your needs.

Make OTU table

This will take a long time for most datasets. If you have tens and tens of thousands (or more) OTUs then I recommend messing around with filtering, clustering, and so on to see if you can get that number to be a bit lower. Making an OTU table with this many OTUs will just take too long and be really hard to analyze. Run time can be drastically cut if multithreading is maximized. I normally run 32 cores, but you could request a node with more cores (multi-node processing is not currently supported with vsearch).

We use original, merged but not filtered data to align to the OTU file. This lets one salvage a few extra reads. If speed is pressing then one could just use the filtered reads and probably would not lose many reads. Make OTU tables for both 16s and ITS data, separately. Note, all these calls to sed reformat the read headers so that the OTU table is sensible and has fields for each sample.

```
cat *16S*merged.fq | sed -e 's/\s/_/g' | sed -e 's/^>16/>rna16/’ | sed -e 's/-/_/g' |

 vsearch --fastx_filter - --fastaout merged16S.fa

vsearch --usearch_global merged16s.fa --db zotus_nonchimeric16S.fa --otutabout 'otutable16S' --id 0.99 --
threads N
```

Could continue the piping, of course, but I was trying to look at intermediate files and catch problems. If you make intermediate files, then delete them when finished to save storage.

Call taxonomy

`vsearch -sintax ITS_OTUS.fa -db /project/microbiome/ref_db/unite4_02_20.fa -tabbedout OUT.sintax -strand both -sintax_cutoff 0.8` --threads N

Use greengenes or Silva for 16S. I have downloaded a variety of commonly used databases into: `/project/microbiome/ref_db` Please update these databases as needed. I will try and do this as well.

If like me (@ Félix Brédoire) and @ Seifeddine Ben Tekaya you have had headaches trying to format the output of `vsearch -sintax`, you can try this.

Suggestions for further work

The coligo table will have internal standard (ISD) reads and you can use that information. However, you will also have ISD reads in your OTU table. You probably will also have plant reads, various nontarget eukaryotes, and sequences that cannot be placed taxonomically. Consider identifying those sequences and summing or removing them as you see fit. Here is an example of how to search OTUs for the ISD:

```
vsearch --usearch_global OTUS.fa --db /project/microbiome/ref_db/synthgene.fasta --blast6out OUT --id 0.8
```

One could use this information to combine or delete ISD otus.

I made a simple script (rowSummer.R) that combines specified OTUs through summing, appends sums as a row to the bottom of an OTU table, and removes the OTUs from that table. Feel free to use it, but please check your results to make sure they are sensible! Check the script out to learn more about using it.

```
Rscript ~/scripts/rowSummer.R otutableITS_plantISD_sum AstragalusMatchesITS "ASLE" "otutableITS_plantISD_sum"
```