Bioinformatic Pipeline For Metabarcoding (OBITools And Friends)

Daniel Marquina (daniel.marquina@nrm.se / danielmarquinahz@gmail.com / @metagusano (Twitter) / metagusano.github.io)

Greetings, be welcomed to the second part of the metabarcoding adventure: the bioinformatic processing (just when you thought you were done with the hard part...).

Gate 1: Installation of the software.

Let's start easy by installing the software that we are going to be using.

1.1 OBITools. A very complete package specifically designed for eukaryotic metabarcoding pipelines. It requires Python 2.7. It is downloaded using the script `get-obitools.py` (python script available here: https://pythonhosted.org/OBITools/welcome.html, or I can share it).

```
$ python get-obitools.py
```

This will automatically download OBITools and create a directory with the latest version of OBITools. Inside there is the executable obitools *that activates the environment for the different scripts to work. This executable should be copied (not moved) to the usr/local/bin/ directory or to a directory that is easily accesible if you are working on a server without the permission to modify the /usr/local/bin, for example ~/bin/ directory. That way it can be activated at any time.*

1.2 ecoPCR. Part of the OBITools universe. It can be obtained here: https://git.metabarcoding.org/obitools/ecopcr/wikis/home. It has to be decompressed and installed.

```
$ tar -zxvf ecopcr-*.tar.gz
$ cd ecoPCR/src/
$ make
```

This will generate a series of executables in the scr/ directory that can be copied to usr/local/bin/ directory or ~/bin/ directory for convenience: `ecoPCR`, `ecofind`, `ecogrep`.

1.3 OwiTools. Set of R scripts that complement other programs or convert file formats. Written by Dr. Owen Wangensteen, can be accessed here: https://github.com/metabarpark/R_scripts_metabarpark. The script `retrieve_BOLD.R` is an older version of the `get_Reference_Database_from_BOLD.R` in his github page. I prefer `retrieve_BOLD.R`, but you are free to look into the updated one and adapt the pipeline accordingly. Once the scripts are downloaded they can be copied to the usr/local/bin/ directory and converted into executables. The files `family_to_order.csv` and `order.complete.csv`, compiled by Owen W. are also part of the package. If not found online, I can provide them.

```
$ chmod +x retrieve_BOLD.R
```

repeat with all owi_ scripts and copy them to the to usr/local/bin/ directory or ~/bin/ directory. Don't copy `family_to_order.csv` *and* `order.complete.csv`*, later we will move them somewhere else.*

1.4 My scripts: **refine_MOTU_table** and **BOLD_identities**. The first is a single script that I wrote to make a final curation of the MOTU table to remove potential false MOTUs and making the table more readable. The second is an script to sending the sequences to BOLD and obtaining the taxonomic identification. They can be accessed: https://github.com/metagusano/metabarcoding_scripts. They are R scripts in executable form that can be copied to usr/local/bin/ or ~/ directory.

1.5 SWARM. It will be used for clustering. It can be found in https://github.com/torognes/swarm.

```
$ git clone https://github.com/torognes/swarm.git
$ cd swarm/src/
$ make
```

Similarly, the executable `swarm` *can be copied to the usr/local/bin/ or ~/bin/ directory.*

1.6 VSEARCH. It will be used for chimera removal, and other steps during the pipeline. It can be found in https://github.com/torognes/vsearch.

```
$ git clone https://github.com/torognes/vsearch.git
$ cd vsearch
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

This will automatically generate an executable in the usr/local/bin/ directory. If you don't have permission, omit the `sudo` *command. The executable can be copied to ~/bin/ directory.*

1.7 USEARCH. For alternative chimera removal and strict denoising. It can be found and downloaded free in https://www.drive5.com/usearch/download.html. It should be copied to our usr/local/bin/ or ~/bin/ directory and converted into an executable.

```
$ chmod +x usearch
```

1.8 FastQC. It can be downloaded from https://www.bioinformatics.babraham.ac.uk/projects/download.html. The folder contains the executable fastqc that should be copied to the usr/local/bin/ or ~/bin/ directory.

1.9 Cutadapt. Requirements: Python 2.7, as it is needed for download and installation. It will generate an executable directly in our usr/local/bin/ or ~/bin/ directory. Documentation: https://cutadapt.readthedocs.io/en/stable/index.html.

```
$ pip3 install --user --upgrade cutadapt
```

1.10 R packages. Most packages are available at CRAN, but not all. Before installing them it is necessary to set-up the downloading machinery. Don't do this if you don't have system permission.

```
 $ sudo apt-get update
 $ sudo apt-get install -y libssl-dev
 $ sudo apt-get install -y libcurl4-openssl-dev
 $ sudo apt-get install -y libxml2-dev
```

Then, we go to R and install the packages. It is important to follow this order, as the latter can't be installed without the former.

```
 > install.packages(c("CHNOSZ", "optparse"))
 ### if an older version of R < 3.5 do
 > source("https://bioconductor.org/biocLite.R")
 > biocLite()
 > biocLite("Biostrings")
```

```
##


### if a newer version of R > 3.5 do
> install.packages("BiocManager")
> BiocManager::install("Biostrings")
##
> install.packages(c("devtools", "dplyr"))
> library(devtools)
> install_github("ropensci/bold")
> install_github("cran/XML")
> install_github("tobiasgf/lulu")
```

And now everything is ready for some awesome science!

Gate 2: Quality filtering and demultiplexing.

The beginning of this gate is going to depend on the method employed for library preparation and multiplexing and on the system of delivering the sequences from the sequencing facility. If the library was prepared by a two-step or one-step-with-phusion-primers PCR method in which the multiplexing tag or index is contained within the Illumina adapter, we will follow the **blue line**. If the library was prepared by enzymatic ligation in which the multiplexing tags are attached at the beginning of the primers, we will follow the **red line** and you have to do steps 2.1 to 2.8 for each library file in parallel and assign a differencing marker to file names (1, 2, 3... or A, B, C...). We will assume that we are working with COI metabarcoding targeting platyhelminths form coral reef substrates, amplified with Leray & Geller's primers.

2.1 You will have received a lot of fastq files from the sequencing facility, twice as many as the number of samples that you have multiplexed in your run (for each sample you will have R1 and R2). The first step is to rename all your files with the sample name and the read number.

```
$ mv P13455_1001_S1_L001_R1_001.fastq turtlepoint_R1.fastq
```

where P13455_1001_S1_L001_R1_001.fastq *is the name of the file I received from the sequencing facility and* turtlepoint *is the name of my sample.*

2.2 The next step is to add an attribute to all the sequences in that file indicating the pertinence to that sample. This is necessary, as we will later merge all reads in a single file. For that we will use a loop that will contain an OBITools command, so we have to activate the environment before running it.

```
$ obitools
$ for file in *_R1.fastq; do
   sample=${file#*dir/}
   sample=${sample%_R1.fastq}
   obiannotate -S sample:${sample} ${sample}_R1.fastq > ${sample}_R1.annotated.fastq
   obiannotate -S sample:${sample} ${sample}_R2.fastq > ${sample}_R2.annotated.fastq
done
$ cat COI*_R1_annotated.fastq >> COI_reefworms_R1.fastq
$ cat COI*_R2_annotated.fastq >> COI_reefworms _R2.fastq
```

personally, I always write loops in scripts rather than in the command line directly, but it is up to you. Don't forget to activate OBITools before start running the loop!

2.3 Now that all reads are together and they have been assigned to samples, we can start the quality control. For that, we will do a FastQC analysis to both files.

```
$ fastqc COI_reefworms_R1.fastq
```

with the .html file produced we can check in an internet browser the quality of the sequences. The most important part is the "quality per base" graph. We have to trim our sequences to preserve only good quality bases. Those regions in which the yellow boxplot passes to the orange area of the plot should be cut away. Let's say in our case is 290 and 270 for R1 and R2 respectively.

2.4 We trim the sequences to the desired length.

```
$ obicut -e 290 COI_reefworms_R1.fastq > COI_reefworms_R1.290.fastq
$ obicut -e 270 COI_reefworms_R2.fastq > COI_reefworms_R2.270.fastq
```

2.5 Now that there are only good sequences, we merge both reads from the same sequence, and we separate those that have a good overlap (Phred score > 40) form the ones with low quality pairing.

```
$ illuminapairedend -r COI_reefworms_R2.270.fastq COI_reefworms_R1.290.fastq |
obiannotate -S goodali:'"Good_COI_reefworms" if score>40.00 else "Bad_COI_reefworms"'
| obisplit -t goodali
```

the flag -r *is used when merging two files (it indicates 'reverse'), and the file with the reads R2 should go first. The command* `obisplit` *will generate two files named* `Good_COI_reefworms.fastq` *and* `Bad_COI_reefworms.fastq`*. We can forget about the latter and proceed with the good one.*

2.6 Our sequences are now the entire barcode. But the primers we used to amplify it have been sequenced as well, so we should remove it or it can mess with the MOTU clustering or the taxonomic assignment later on. For that we use cutadapt.

```
$ cutadapt -g GGWACWGGWTGAACWGTWTAYCCYCC -a GRTTYTTYGGICAYCCIGARGTITA -o
COI_reefworms.filtered1.fasta Good_COI_reefworms.fastq
$ cutadapt -g GGWACWGGWTGAACWGTWTAYCCYCC -a GRTTYTTYGGICAYCCIGARGTITA -o
COI_reefworms.filtered2.fasta COI_reefworms.filtered1.fasta
```

`cutadapt` *will look for the forward primer at the beginning of the sequence with the flag* -g*, and for the reverse primer at the end of the sequence with the flag* -a *and will trim them out. Note that the reverse primer has been incorporated to the sequence in the forward direction, so we need to write the reverse complementary of the primer we ordered for the PCR. To make sure we remove the reverse primer, we run* `cutadapt` *twice (sometimes it misses it). The* -o *option specifies the output, which we have indicated that we want in fasta format, not in fastq, as we have already done the quality filtering and we don't care about Phred scores any more.*

2.7 For the next step, we will need another attribute of the sequence: sequence length. So we have to add it to the sequence headers.

```
$ obiannotate --length COI_reefworms_filtered2.fasta > COI_reefworms.noprimers.fasta
```

JUMP TO POINT 2.8 (BLACK) NOW, DON'T GET CONFUSED!

2.1 The first step is to rename your files with the R1 and R2 reads from whatever complicate name you receive from the sequencing facility to something that you can identify.

```
$ mv P13455_1001_S1_L001_R1_001.fastq COI_reefworms_R1.fastq
$ mv P13455_1001_S1_L001_R2_001.fastq COI_reefworms_R2.fastq
```

where P13455_1001_S1_L001_R1_001.fastq *is the name of the file I received from the sequencing.*

2.2 Then we begin the quality control by running a FastQC analysis on both files.

```
$ fastqc COI_reefworms_R1.fastq
```

with the .html file produced we can check in an internet browser the quality of the sequences. The most important part is the "quality per base" graph. We have to trim our sequences to preserve only good quality bases. Those regions in which the yellow boxplot passes to the orange area of the plot should be cut away. Let's say in our case is 290 and 270 for R1 and R2 respectively.

2.3 To trim the sequences we will use one of the OBITools scripts, so first we have to activate the environment and then do to the trimming.

```
$ obitools
$ obicut -e 290 COI_reefworms_R1.fastq > COI_reefworms_R1.290.fastq
$ obicut -e 270 COI_reefworms_R2.fastq > COI_reefworms_R2.270.fastq
```

2.4 Now that there are only good sequences, we merge both reads from the same sequence, and we separate those that have a good overlap (Phred score > 40) form the ones with low quality pairing.

```
$ illuminapairedend -r COI_reefworms_R2.270.fastq COI_reefworms_R1.290.fastq |
obiannotate -S goodali:'"Good_COI_reefworms" if score>40.00 else "Bad_COI_reefworms"'
| obisplit -t goodali
```

the flag -r *is used when merging two files (it indicates 'reverse'), and the file with the reads R2 should go first. The command* `obisplit` *will generate two files named* `Good_COI_reefworms.fastq` *and* `Bad_COI_reefworms.fastq`*. We can forget about the latter and proceed with the good one.*

2.5 With the good-paired sequences, it's time to convert the fastq into fasta, and to demultiplex the reads (assigning each read to its correspondent sample, based on the tags used during the PCR). For that we need a tab or space separated file in .tsv or .txt format where we specify the information regarding the sequences of the sample tags and the primers. The file should look like this:

```
#exp      sample tags   forward_primer        reverse_primer
Reef      turtle_point  TGCAC:CTGAC   GGWACWGGWTGAACWGTWTAYCCYCC TANACYTCNGGRTGNCCRAARAAYCA
Reef      tongue_reef   TGCAC:TCAGC   GGWACWGGWTGAACWGTWTAYCCYCC TANACYTCNGGRTGNCCRAARAAYCA
Reef      lizard_island TGCAC:ATCGC   GGWACWGGWTGAACWGTWTAYCCYCC TANACYTCNGGRTGNCCRAARAAYCA
Reef      Empty1        TGTCA:TAATG   GGWACWGGWTGAACWGTWTAYCCYCC TANACYTCNGGRTGNCCRAARAAYCA
...
```

where we list the sample and the associated tags in the forward and reverse primer (tagF:tagR) and the PCR primers. Note that the deoxyinosines (I) in the reverse primer have been replaced with Ns, as obitools does not accept other than IUPAC coding. It is a good idea to include some tag combinations that were not used during the sequencing (e.g. last line with Empty1) to later calculate a proportion of the sequences that might have been assigned to the wrong sample due to tag jumping or sequencing errors.

```
$ ngsfilter -t ngsfilter_COI.tsv --fasta-output -u unidentified.fasta
Good_COI_reefworms.fastq > COI_reefworms.noprimers.fasta
```

-u *means that every sequence that can not be assigned to any sample, will be dumped in the* `unidentified.fasta` *file. In* `COI_reefworms.noprimers.fasta` *we have all the sequences assigned to their samples, with the primer and tag sequences trimmed out.*

2.6 Now, if you have several libraries (let's say 3 different R1 and R2 fastq files) you have probably been doing every step in triplicates (you should have, at least). This is the point where you can pool all your sequences together, as now the all have in the header the information about what sample they belong to.

```
$ cat COI_reefworms*.noprimers.fasta >> COI_reefworms_pooled.noprimers.fasta
```

* *means that all three files* `COI_reefworms_A.noprimers.fasta`, `COI_reefworms_B.noprimers.fasta` *and* `COI_reefworms_C.noprimers.fasta` *will be incorporated to the file, and we use* >> *instead of* > *so the information of each original file is added to the new file instead of overwriting it.*

WE JUMP FROM POINT 2.6 TO POINT 2.8 NOW, DON'T GET CONFUSED!

2.8 The next filtering step will be removing those sequences that do not fit in our expected sequence length, as they would probably be pseudogenes, PCR/sequencing errors, unspecific amplification or off-target organisms' barcodes (bacteria, etc.). This doesn't mean that everything that passes the filter is perfect quality, biologically meaningful or only target organisms' barcodes. We can apply a low stringency margin of around 3 % for protein-coding genes, but for ribosomal genes the margin should be wider. With Leray & Geller primers the amplicon should be 313 bp long.
It is better to check first the frequency distribution of sequence length in case anything weird is happening (too short or too long sequences)

```
$ obistat -c seq_length COI_reefworms_pooled.noprimers.fasta
```

this will give us a table of the number of sequences for each sequence length value that we can copy or export and plot in R, or excel or whatever.

2.9 With the interval of length we observed in the graph, we can proceed to do the filtering.

```
$ obigrep -p 'seq_length>303' -p 'seq_length<323' -s '^[ACGT]+$'
COI_reefworms.noprimers.fasta > COI_reefworms.length.fasta
```

the argument -s '^[ACGT]+$' *indicates that it will only keep sequences composed of A, C, T, G, no Ns or any other ambiguous bases, while* -p *indicates that a Python expression is going to follow.*

2.10 We can now trust (or hope) that the sequences that remain are the ones we are interested in. Only to reduce the computation effort, we will collapse all the sequences that are identical, but we will keep the information of how many copies of each there are in the different samples.

```
$ obiuniq -m sample COI_reefworms.length.fasta > COI_reefworms.unique.fasta
```

the flag -m *is the argument for keeping the sample information and abundance.*

2.11 Also for convenience, we will shorten the sequence names, because M01320:274:000000000-CJBFL:1:2107:12171:1380 it's a bit tricky to manage, while COI_001 is better.

```
$ obiannotate --seq-rank COI_reefworms.unique.fasta | obiannotate --set-identifier
'"COI_%d"%seq_rank' > COI_reefworms.shortnames.fasta
```

--seq-rank *will generate a new attribute in the header of the sequence based on its position in the file (1, 2, 3…), and then,* --set-identifier *will rename the sequence based on that number.*

2.12 To finish this gate, we transform the last file into a tab file that we will keep untouched until later. This file will contain the sequence identifier and the read abundance in every sample.

```
$ obitab -o COI_reefworms.shortnames.fasta > COI_reefworms.tab
```

Getting this far is a success on itself, so go grab a cup of coffee and relax for five minutes. You've earned it!

Gate 3: MOTU clustering.

At the beginning of this gate we have a file with sequences that are ‘technically’ valid. This means that they have a good sequencing quality, the length they are supposed to have and we are keeping track of the precedence and the abundance of each unique sequence. But that doesn’t mean that all of them are *biologically* valid. There are several processes that can generate false sequences (PCR errors, sequencing errors, pairing errors, etc.), known as chimeras. We are going to follow here the MOTU (Molecular Operational Taxonomic Unit) strategy, in which we will collapse sequences that are similar in clusters (with a threshold that we will choose ourselves depending on the marker and the target organisms), considering all of the sequences in a cluster as natural variants belonging to the same species. Other metabarcoding pipelines skip this step and use the unique sequences, or Exact Sequence Variants (ESV), for taxonomic assignment. Ideally, at the end of this gate, we will have as many MOTUs as species in our samples. Alternative steps are in the **orange line** and optional steps in the **brown line**.

3.1 The first step of this gate is to remove the chimeras. For that we will use VSEARCH, so we need to transform the file format to something that VSEARCH can read. We do that with one of the OwiTools scripts and text edition, and we order the file in decreasing number of reads to help the deschimeration.

```
$ owi_obifasta2vsearch -i COI_reefworms.shortnames.fasta -o
COI_reefworms.vsearch.fasta
$ sed -i -e 's/[[:space:]].*;size/;size/g' COI_reefworms.vsearch.fasta
$ vsearch -sortbysize COI_reefworms.vsearch.fasta -output
COI_reefworms.vsearch.sorted.fasta -minsize 1
```

3.2 With the file in the correct format, we can now remove the chimeras.

```
$ vsearch --uchime_denovo COI_reefworms.vsearch.sorted.fasta --sizeout --nonchimeras
COI_reefworms.nonchimeras.fasta --chimeras COI_reefworms.chimeras.fasta --uchimeout
COI_reefworms.uchimeout.txt
```

--sizeout *indicates that it will take read numbers into account when deciding whether a read is a real sequence or a chimera. The file that matters most is* COI_reefworms.nonchimeras.fasta*, which has the real sequences, while* COI_reefworms.chimeras.fasta *contains the chimeras, and* COI_reefworms.uchimeout.txt *contains the register of all the information of the sequences analyzed, being determined chimeras or not.*

3.2.1 Alternatively, a more stringent denoising and des-chimeration can be done using UNOISE3. It takes very long, but is stricter, which can be advantageous if we are planning on doing some type of haplotyping with the data. Otherwise, the small errors below species level left by VSEARCH can be removed during clustering or post-clustering curation.

```
$ usearch -unoise3 COI_reefworms.vsearch.sorted.fasta -minsize 1 -unoise_alpha 5 -
zotus COI_reefworms.zotus.fasta -tabbedout COI_reefworms.unoise.txt
```

the argument -unoise_alpha *will determine the stringency of the denoising. The higher it is, the more strict in assigning errors. The default value is 2, but some metabarcoding-haplotyping studies have shown that a value of 5 gives best results. The only problem with this command is that requires lots and lots of RAM memory and that the resulting file* COI_reefworms.zotus.fasta *has lost the sequence identifiers, so some edition is necessary to recover the original sequence names (see below).*

3.2.2 Although this could be thrown in a script, it is quite straight forward, so we can do it directly in R.

```
> library(Biostrings)
> all_seqs <- readDNAStringSet(COI_reefworms.vsearch.sorted.fasta)
> zotus <- readDNAStringSet(COI_reefworms.zotus.fasta)
> length(which(all_seqs %in% zotus))
> length(zotus)
```

at this point, if `length(which(all_seqs %in% zotus))` *is not the same as* `length(zotus)` *is that we messed it up somewhere and it should be best to redo the previous step. If it is we finish this step:*

```
> denoised <- all_seqs[c(which(all_seqs %in% zotus))]
> writeXStringSet(COI_reefworms.zotus.names.fasta)
```

and it will be COI_reefworms.zotus.names.fasta *the file we will use in point 3.3.*

3.3 The next program can deal with truncated fasta files, but the manual states that it works better with single line fasta files, so we have to transform the 'nonchimeras' or the 'zotus' into the latter type. For that we use the following command:

```
$ awk '/^>/{print (NR==1)?$0:"\n"$0;next}{printf "%s", $0}END{print ""}'
COI_reefworms.nonchimeras.fasta > COI_reefworms.nonchimeras_singleline.fasta
```

3.4 Now we will do one of the most important steps in the pipeline. We can customize it in several ways, but this will affect the results greatly, so one must be careful with it and have a justification for each change.

```
$ swarm -d 13 -z -t 1 -o COI_reefworms_swarm13_output -s COI_reefworms_swarm13_stats -
w COI_reefworms_swarm13_seeds.fasta COI_reefworms.nonchimeras_singleline.fasta
```

-d *is the maximum distance, expressed in <u>number of nucleotides</u> (NOT percentage!), for two sequences to be considered part of the same MOTU; -z means that the input file is in vsearch format;* -t *is the number of cores you are using (depending on you computational power). The* _output *and* _stats *files are very similar to each other, and basically tell you how many MOTUs you have, and which sequences belong to each one of them. The* _seeds.fasta *file has only the main sequence of each MOTU (also called centroid or head in other programs), and it will be used for the taxonomic assignment.*

If it's your first time working with the taxonomic group or/and the marker, it is recommendable to do several swarms with different distance levels, like 4-5 points up and down the expected barcoding gap. For instance, for this marker (313 bp long COI) it would be from 5 to 15. Then, we plot the number of MOTUs generated against the distance and we select the ones in which the number of MOTUs keeps more or less constant, because that will be indicating the real barcoding gap. Let's assume in this case that it's in 13 (which is around 96 -97 % similarity).

3.5 Combining `COI_reefworms_swarm13_output` and `COI_reefworms.tab` (the one we created at the end of Gate 2) we can calculate the read abundance of each MOTU in each sample, by pooling together the read numbers of all the sequences belonging to the same MOTU, and write it in a csv file. We do this with another script from OwiTools. This script can be edited in one of the first lines to only include MOTUs with a total count of 1 read (*i.e.* MOTUs consisting on 1 sequence with only 1 read across all samples), in this line: `min_reads <- 2`.

```
$ owi_recount_swarm COI_reefworms_swarm13_output COI_reefworms.tab
```

the output of this will be a file called COI_reefworms_swarm13_output.counts.csv.

Consequently, we must select from the output fasta file from SWARM the size >= 2 centroid sequences:

```
$ sed -i 's/;;size=/ size=/g' COI_reefworms_swarm13_seeds.fasta
$ obigrep -p 'size>1' COI_reefworms_swarm13_seeds.fasta >
COI_reefworms_seeds.nosingletons.fasta
```

3.6 It is known that the clustering is not perfect, and we can find different MOTUs that actually belong to the same species, getting high levels of MOTU redundancy (getting many MOTUs with the same taxonomic assignment in the same sample) we can use LULU algorithm, that identifies redundant MOTUs based on similarity and co-occurrence. This is particularly interesting for pseudogenes, which might have been amplified during the PCRs. So we need the fasta with the seed sequences and the table with the abundances in each sample. Before the actual curation, we have to prepare the data first and generate a table with the similarity values between MOTUs' seeds.

```
$ sed -i 's/ size=\([0-9]\+\);//g' COI_reefworms_seeds.nosingletons.fasta
```

this will remove the ";size=N;" part of the identifier of the MOTU to get a cleaner file.

```
$ vsearch --usearch_global COI_reefworms_swarm13_seeds.fasta --db
COI_reefworms_swarm13_seeds.fasta --self --id .84 --iddef 1 --userout
COI_reefworms.match_list.txt -userfields query+target+id --maxaccepts 0 --query_cov .9
--maxhits 10
```

COI_reefworms.match_list.txt *is the file with the similarity table that LULU will use for deciding if a MOTU is an error or not.* --id .84 *indicates that 84 % is the minimum similarity value considered;* --maxhits *is the maximum number of MOTUs whose similarity is recorded for each MOTU (descending value).*

```
$ cut -d ";" -f1,A-B COI_reefworms_swarm13_output.counts.csv >
COI_reefworms.counts_LULU.txt && sed -i 's/;/\t/g' COI_reefworms.counts_LULU.txt
```

with this command we are transforming the output.counts.csv form the owi_recount step into a simpler table with only id and the reads/sample, which is the only info that LULU considers. The argument -f 1, A-B, *indicates to print only the columns 1 (id) and A-B (samples). You have to substitute A and B for the first and last sample column of your table: if you have 10 samples starting in column 15, then A=15, B=24.*

3.7 With all these files ready we move to R, which can be used from the command line.

```
$ R
> library(lulu)
> matchlist <- read.table("COI_reefworms.match_list.txt", header=FALSE, as.is=TRUE,
stringsAsFactors=FALSE)
> otutab <- read.csv("COI_reefworms.counts_LULU.txt", sep='\t', header=TRUE,
as.is=TRUE, row.names = 1)
> curated_result <- lulu(otutab, matchlist)
```

the curated table will be a part of the object curated_result *and can be written in a file for the last step by typing:*

```
> write.csv(curated_result$curated_table,"COI_reefworms.LULU.curated.csv")
```

And we keep that file warm and comfy, until the last Gate.

Gate 4: Reference database and taxonomic annotation.

It is usually said that a metabarcoding study is only as good as the reference database that is used on it. While this is not totally true, having a good reference database is going to produce much better results, as we are going to be able to identify our MOTUs. There are times when this will be not as important, for instance, if you are analysing the diversity of two areas with different level of perturbation, or community differences based on an environmental factors. But in other cases, such as surveys to detect invasive species, or diet assessment, it is of critical importance to have as complete a database as possible. In this Gate we are going to retrieve the sequences from public repositories (EMBL and BOLD, but it can also be done from GenBank) and transform them into a database formatted so that OBITools can use to compare with the sequences we have obtained from our study. The specifics of this database will change with the group and marker YOU are working with, but in this example case it will be COI of flatworms. As an alternative (**orange line**), only valid for metazoan metabarcoding, you can use the script *BOLD_identities* to query your sequences directly to BOLD.

4.1 First of all, we need the taxonomy information to build the database, and that will be in the NCBI taxonomic repository that has to be downloaded and converted into a format we can use. Let's assume we are analysing our data in in June 2020.

```
$ mkdir TAXO/ && cd TAXO/
$ wget -m ftp://ftp.ncbi.nlm.nih.gov://pub/taxonomy/taxdump.tar.gz
$ mv ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz .
$ gunzip *
$ tar -xvf taxdump.tar
$ rm -rf ftp.ncbi.nlm.nih.gov/
$ rm taxdump.tar
$ cd ../
$ obitaxonomy -t ~/TAXO/ -d taxonomy_june20
```

`obitaxonomy` *will convert all the files in our directory* TAXO *into the taxonomy database,* taxonomy_june20*, that we will later use to link sequences and species information to create the reference database.*

Copy the files `family_to_order.csv` and `order.complete.csv` from OwiTools in this directory, as they will be used later on.

4.2 The sequences from EMBL can be downloaded directly using the ftp directory. The ENA (European Nucleotide Archive) makes publicly available 4 releases every year, which is great because this way we make sure to have the latest and most complete data out there. These releases include various types of data, of which we are interested in the STD: the annotated sequences. There are sequences of all organisms, and it's organized in several "taxonomic" divisions: env (environmental), fun (fungi), hum (human), inv (invertebrates), mam (mamals excluding humans or rodents), mus (mice), phg (phages), pln (plants), pro (prokaryonts), rod (rodents excluding mice), syn (synthetic), tgn (transgenic), unc (uncultured), vrl (virus) and vrt (vertebrates excluding mamals). We can download it complete or just the division we are interested in, in this case INV (flatworms). Let's assume the latest release is the r140.

```
$ mkdir REF/ && cd REF/
$ wget -m ftp://ftp.ebi.ac.uk/pub/databases/embl/release/std/rel_std_inv_*_r140.dat.gz
$ mv ftp.ebi.ac.uk/pub/databases/embl/release/std/*.gz .
$ rm -rf ftp.ebi.ac.uk/
$ gunzip *
$ cat rel_std_inv*.dat >> EMBL_inv_r140.dat
$ obiconvert --embl --fasta-output EMBL_inv_r140.dat > EMBL_inv_r140.fasta
```

in this EMBL_inv_r140.fasta *we have all the sequences of every gene of every species of invertebrate that has been sequenced and uploaded to GenBank and other sources (NOT BOLD).*

4.3 Downloading the sequences from BOLD can also be done from the command line. For that we will need the script `retrieve_BOLD.R`. But before running it we have to set the taxa names we are interested in, and the directory where the taxonomy dump files downloaded from NCBI are. It should look like this:

```
...
taxdir <- "/path/to/directory/TAXO" # Folder with NCBI taxdump
nodes <- getnodes(taxdir)
nombres <- getnames(taxdir)

taxon <- c("Polycladida", "Catenulida", "Tricladida", "Prolecitophora",
"Macrostomorpha", "Lecitoepiteliatha", "Botrioplanida", "Rhabdocoela", "Proseriata")
#change taxa
logfile <- paste("BOLD_190620","_retrieve_bold.log",sep="")  #change names
...
```

the script will write a fasta file for each of the different taxa we have selected. I wrote all the orders of the free living Platyhelminthes (according to BOLD's taxonomy) instead of just "Platyhelminthes" because sometimes the BOLD API doesn't accept high ranks. The same happens with Insects, for example.

4.4 With all the fasta files we have now from BOLD and EMBL we can build an ecoPCR database.

```
$ obiconvert -d taxonomy_june20 --fasta --ecopcrdb-output Reef_Worms --skip-on-error
*.fasta
```

the argument --fasta *means that the input is in fasta format,* --ecopcrdb-output *is the formatted database, named Reef_Worms, that will contain several files (.ndx, .pdx, .sdx…); some sequences will give problems, so to avoid the whole process stop because of it, we type* --skip-on-error *(we will lose some sequences, but there's nothing we can do). In some versions of OBITools, the output format can be* --ecopcrDB *instead (as in OBITools tutorial), and that can be a source of error too, make sure to type the correct command depending on you version.*

4.5 The next step is to create a simulation of sequencing with the database, using `ecoPCR`. As said before, let's assume we have sequenced our samples with Leray & Geller's primers:
mlCOIintF GGWACWGGWTGAACWGTWTAYCCYCC - jgHCO2198 TAIACYTCIGGRTGICCRAARAAYCA, amplicon size of 313 bp.

```
$ ecoPCR -d Reef_Worms -e 3 -l 303 -L 323 GGWACWGGWTGAACWGTWTAYCCYCC
TANACYTCNGGRTGNCCRAARAAYC > Reef_worms_ref.ecopcr
```

the argument -e 3 *allows for 3 mismatches between primer and template to produce the amplification (better more sequneces than less),* -l 303 -L 323 *is the min and max length allowed of the resulting amplicon. Note that the deoxyinosines (I) in the reverse primer have been replaced with Ns, as obitools does not accept other than IUPAC coding.*

4.6 This file has the sequences of the resulting amplicons if the species retrieved were amplified with the selected primers. It needs to be transformed into fasta format, and the taxonomic rank above family added, so we have all the information possible when doin the taxonomic annotation.

```
$ obiconvert --output-fasta Reef_worms_ref.ecopcr > Reef_worms_ref.fasta
$ obiannotate -d Reef_worms --with-taxon-at-rank=phylum --with-taxon-at-rank=class --
with-taxon-at-rank=subclass --with-taxon-at-rank=order Reef_worms_ref.fasta >
Reef_worms_ref_hightax.fasta
```

the file `Reef_worms_ref_hightax.fasta` *is the one that will be used as reference in the next step.*

4.7 The following step is one of the most critical, together with the SWARM, in the pipeline. Here we use the reference fasta file we have just created with the centroid sequences of each MOTU to obtain the taxonomic annotation of the latter with the *ecotag* algorithm.

```
$ ecotag -d Reef_worms -R Reef_worms_ref_hightax.fasta
COI_reefworms_seeds.nosingletons.fasta > COI_reefworms.ecotag.fasta
```

COI_reefworms.tag.fasta *will have the taxonomic assignment (in the best cases species, but it can be genus, family, order, etc.) in the header. It would contain more MOTUs than the* COI_reefworms.LULU.counts.*csv file, but there is no problem with that, because later on we will only keep the good ones.*

4.8 The tagged file COI_reefworms.ecotag.fasta will contain the taxonomic annotation, but only up to family level, because that is the default for ecotag (that's why we required at least family level for building up the database). But using another script from OwiTools, and with the files family_to_order.csv and order.complete.csv we can add all the taxonomic categories to each MOTU. Before running it, we have to edit it to add the directory where the taxonomy is:

```
dir_taxo <- "/path/to/directory/TAXO/" # Folder with NCBI taxdump
```

```
$ owi_add_taxonomy COI_reefworms.tag.fasta
$ sed -i -e 's/,/./g' COI_reefworms.tag.fasta.annotated.csv
```

this will create the file COI_reefworms.tag.fasta.annotated.csv *with all the information. It is important to edit it to set the correct directory of the taxonomy, the same as in previous scripts. The second command will change the commas for dots in the column referring to the identity to the reference sequence.*

4.1 As an alternative, you can use the scrip BOLD_identities to query the centroid sequences to BOLD. This script uses the API system and not the bold package of R, so the querying doesn't saturate and stop after any number of queries. It will take the centroid sequence of each MOTU, obtain taxonomic identification, and adjust it to the most adequate taxonomic level based on similarity thresholds.

```
$ BOLD_identities -f COI_reefworms_seeds.nosingletons.fasta -t
COI_reefworms_taxonomy.csv
```

where the flag -t *indicates the name of the output table that will contain the taxonomic information and the sequence.*

And we are almost there, just one more Gate and we will have out final dataset!

Gate 5: Combine and refine.

Now we count with two main table files: 1) a file containing the read numbers per sample for each MOTU generated during SWARM, `COI_reefworms_swarm13_output.counts.csv`, or the one corrected by LULU, `COI_reefworms.LULU.curated.csv`, and 2) a file containing the taxonomic information of the centroid sequences of each of the MOTUs, `COI_reefworms.ecotag.annotated.csv` (or `COI_reefworms_taxonomy.csv` if you used the BOLD querying strategy in Gate 4). Combining these two will generate a table with both the abundance in each sample and the taxonomic assignation of the MOTUs. But that table is not perfect and final, as there are some more refinements that will improve the presentation and analysis. At the beginning we will start with the line that included LULU curation, and the line that did NOT include it starts at 5.2 (black).

5.1 First of all, since the counts and the taxonomy tables come from two different software, the formatting is different. We have to make them compatible, and it is just a matter of a little text editing.

```
$ sed -i -e 's/""/"id"/g' COI_reefworms.LULU.curated.csv
$ sed -i -e 's/"//g' COI_reefworms.LULU.curated.csv
$ sed -i -e 's/,/;/g' COI_reefworms.LULU.curated.csv
```

5.2 Now we combine both files with another one of the OwiTools scripts.

```
$ owi_combine -i COI_reefworms.ecotag.fasta.annotated.csv -a
COI_reefworms.LULU.curated.csv -o COI_reefworms.combMOTUs.csv
```

5.2 [Skip this point if you have done LULU curation!] To combine taxonomic and abundance information we will use another OwiTools script.

```
$ owi_combine -i COI_reefworms.tag.fasta.annotated.csv -a
COI_reefworms_swarm13_output.counts.csv -o COI_reefworms.combMOTUs.csv
```

5.3 This script does not, however, transfer all the information of both tables. It leaves out the sequence of the centroid, so we have to edit it in R to add in the sequence from the taxonomy table.

```
> table_abundances <- read.csv("COI_reefworms.combMOTUs.csv", sep=";", header=T)
> taxonomy_sequences <- read.csv("COI_reefworms.ecotag.fasta.annotated.csv", sep=";",
header=T)
> just_sequences <- taxonomy_sequences[,c("id", "sequence")]
> table_abundance_sequences <- merge(table_abundances, just_sequences, by="id")
> write.csv2(table_abundance_sequences, "COI_reefworms.combMOTUs.seqs.csv",
row.names=F)
```

we use here `write.csv2` *rather than write.csv because the former will use " ; " as separator and " ," as decimal point. This is convenient because in later uses,* `R` *will not consider the column* best_identity *as numeric but as factor. Also, it will make possible to use the next and final script.*

5.4 The combined–with–sequences table still contains a lot of low abundance MOTUs product of sequencing errors or chimeras that managed to pass the previous filters. Also, it will probably have many MOTUs with the same species identification that can be collapsed to be considered as a single occurrence. But above all, during the sequencing it is likely that some reads have been assigned to an erroneous sample due to tag switching (if the sample tag was on the Illumina adapter (2-step PCR)) or PCR/sequencing errors (if the sample tag was on the PCR primer (enzymatic ligation)). The way of spotting this is to have included some tag combinations that didn't have any real sample, or positive controls with species that we know were not present in our samples.

We count the total number of reads in the table, and the reads in the following categories:
- reads in the "empty" samples
- reads of the positive species in real samples
-reads of non-positive species in the samples unique to the positive species.

This will give us a number *T*, which divided by the total number of reads will give us a proportion *t* of reads erroneously assigned in average to each sample. Let's say this proportion is 0.004 (0.4 %) of the reads.
Also, we have to decide what is the minimum number of reads in total across all samples for each MOTU to be trusted as a real occurrence (the famous singletons and doubletons). Some people use 1, 2, 5 or 10. Let's say we want at least 10 reads (remember than we have removed singletons in a previous step, so using the value 1 won't do anything).
Finally, we have to decide if we want the different MOTUs assigned to the same species as separate MOTUs or if we want them to aggregate them in the same occurrence. Let's say we want to aggregate them. We must be careful, since due to reference database incompleteness, some MOTUs could be assigned to species level, but with very low similarity to the reference sequence of that species (this happens when we have one reference sequence for a broad taxon, the query sequences will be more similar to that sequence than to any other reference in the database). Then, the aggregation has to consider the similarity value of the MOTU with the reference sequence.
With this in mind, we run the last script

```
$ refine_MOTU_table -i COI_reefworms.combMOTUs.seqs.csv -o COI_reefworms.final.csv -t
0.004 -r 10 -a yes -s 0.96
```

the flag `-i` *indicates input table*, `-c` *output table,* `-t` *is the proportion of mistagged reads,* `-r` *the minimum number of reads for trusting a MOTU and* -a `yes` *indicates that we want to aggregate same-species-determination MOTUs, and* `-s` *that this aggregation will be done only for those MOTUs that have a similarity value of at least 96 % with the assigned reference species. For the default values check* https://github.com/metagusano/metabarcoding_scripts.

And that is all!! You have now your dataset ready for whatever analysis we want to do with it. Congratulations!!!