Metagenomic Amplicon analysis pipeline

Table of Content

Introduction

This workflow describes the analysis of amplicon data, specifically of ribosomal markers such as 16S, 18S and ITS genes.
The goal of this workflow is to create a table of OTU abundances per sample from forward and reverse read fastq files.

> Although this workflow can in general also be used to analyse amplicon sequences of functional genes I recommend to add some additional steps such as, frame shift correction.

Workflow summary

Initial data set

This workflow assumes that your initial data set is:

- demultiplexed into two fastq files per sample: one for forward reads and one for reverse reads
- barcodes, adapters, linkers and primer sequences are already "clipped" (removed) from the reads
- File names of your forward and reverse read fastq files include *R1* and *R2*, respectively.
  Example:
  Forward read fastq file name: **S25_H15_R1_L001.fastq**
  Reverse read fastq file name: **S25_H15_R2_L001.fastq**

> The ending of fastq files may differ, whereas the most common file extension is ***fastq*** some sequencing providers and tools may also use the ending ***fq***.
>
> *However, the file extension does no affect the file content!*

Step 1: Concatenate reads

Many amplicon sequencing projects include many different samples for which the forward and reverse reads have to be joined into longer fragments. By joining all forward fastq files into one file and all reverse reads into another we don't have to run the joining for each sample separately but can do this in one step.

To concatenate the forward and reverse read files we use the linux command *cat*. It takes two or more files as input and prints their content to the command line, which we then re-direct into another file using the re-direction operator ">".

To concatenate all files in a directory that include "R1" in the file name and write them into a new file called "forward.fastq" type

```
cat *R1* > forward.fastq
```

Some characters on Unix command line and in certain programming languages have special meanings. One of them is the asterisk which means "*any character*".
Thus, the command above in English means:

Concatenate all files named as follows: any character(s) followed by R1 followed by any character(s) and redirect the resulting output into file forward.fastq

Do the same with all files including "R2" in the file name and create a new file called "reverse.fastq"

```
cat *R2* > reverse.fastq
```

Make sure that only forward read files include "R1" in the name and only reverse read file names include "R2". Otherwise you will mix forward and reverse reads and your analysis will produce wrong results!

How to:

To check what files in a directory include a certain pattern use the linux command "ls" (short for *list*).
For example, to check which files in the current directoryinclude "R1" you can use a similar syntax as for the *cat* command above:

```
$>ls *R1*
```

Step 2: Join read pairs to fragments

Often paired-end reads of amplicon sequencing projects overlap for a certain number of bases, especially if the region of interest is longer than the average length of a read. For example, if we want to sequence a DNA fragment of 595bp length we will not be able to do this with a single Illumina read because the longest reads currently produced by Illumina MiSeq are 300bp. To be able to sequence the complete fragment we can sequence the first 300bp of the fragment in 5' to 3' direction (forward) and other 300bp of the reverse direction 3' to 5'. This will lead to an overlap of 5 nucleotides which enables us to join the reads to one 595bp long fragment.

To join all forward and reverse reads use the read joiner FLASH by calling

```
flash --cap-mismatch-quals forward.fastq reverse.fastq
```

This command will create several files:

- out.extendedFrags.fastq = a fastq file of the joined fragments
- out.notCombined_1.fastq = a fastq file of all forward reads that could not be joined with a corresponding reverse read
- out.notCombined_2.fastq = a fastq file of all reverse reads that could not be joined with a corresponding forward read
- out.hist = a numeric histogram of the length of the joined fragments
- out.histogram = a visual histogram of the length of the joined fragments

The file we're going to use for the rest of the analysis is the *out.extendedFrags.fastq* file.

A word of advice: Check your files!

It is good practice to check your intermediate result files after each step to make sure that everything went ok. Otherwise you might end up with cryptic or wrong results and don't know which step went wrong.

An easy way to check the number, length and quality of sequences will be described in the next step.

How many reads were actually joined?

In this step it would be wise to check how many of your sequences actually joined into fragments: Count the reads in your forward or reverse file and compare it with the number of joined fragments.

The percentage of reads that didn't join can vary from a few percentage to >50% depending on read quality or how many reads actually overlapped. A common mistake is over-trimming of reads BEFORE joining. In that case sometimes the overlapping ends of one of the reads is removed due to low quality which makes subsequent joining impossible.

Step 3: Filter reads/fragments

We'll use the widely known amplicon analysis framework mothur to filter fragments by length and different quality scores.

To do this we first have to convert our fastq file into a fasta file and a quality file using mothur's fastq.info command.

```
# start mothur
mothur

# convert fastq to fasta and qual file
mothur> fastq.info(fastq=out.extendedFrags.fastq)
```

> In many programming languages the hashtag *(#)* indicates a so called *comment*, i.e., the computer ignores everything following the hashtag.

This creates multiple files, among others *out.extendedFrags.fasta* and *out.extendedFrags.qual,* which contains the quality lines of the fastq file.

Using mothur's *summary.seqs* we can get an overview over the number of our fragments, the length distribution, number of ambiguous bases and other stats.

```
mothur > summary.seqs(fasta=out.extendedFrags.fasta,processors=8)

Using 8 processors.

         Start   End      NBases   Ambigs     Polymer NumSeqs
Minimum:    1     37      37    0       2     1
2.5%-tile:  1     148      148      0      4     428163
25%-tile:   1     559      559      0      6     4281628
Median:     1     560      560      0      6     8563255
75%-tile:   1     560     560      0      6     12844882
97.5%-tile: 1     561      561      0      8     16698347
Maximum:    1     592      592      128      149     17126509
Mean:     1     542.969  542.969  0.107544    6.2282
# of Seqs:    17126509
```

In this example you can see that the majority of sequences are between 559bp and 561 in length, contain 0 ambiguous bases and contain 6 homopolymers.

Mothur's trim.seqs can now be used to remove all sequences we don't want. For this particular example it might be a good idea to remove sequences that are shorter or longer than 559-561bp, contain more than 6 homopolymers and contain any ambiguous bases.

```
mothur >
trim.seqs(fasta=out.extendedFrags.fasta,minlength=559,maxlength=561,maxh
omop=6,maxambig=0,processors=8)
```

This will produce two files:

1. out.extendedFrags.trim.fasta
   This fasta file contains all sequences that fulfilled our trimming requirements, i.e., are "good" sequences
2. out.extendedFrags.scrap.fasta
   This fasta file contains all sequences that did not make it through trimming, i.e., the "bad" sequences

Let's get some statistics on our good sequences using summary.seqs

```
mothur > summary.seqs(fasta=out.extendedFrags.trim.fasta)

Using 8 processors.

          Start   End      NBases     Ambigs     Polymer     NumSeqs
Minimum:     1     559      559      0      3      1
2.5%-tile:     1     559      559      0      5      314315
25%-tile:    1     560      560      0      6      3143144
Median:      1     560      560      0      6      6286288
75%-tile:     1     560      560      0      6      9429432
97.5%-tile:     1     561      561      0      6      12258261
Maximum:     1     561      561      0      6      12572575
Mean:      1     559.866     559.866     0      5.8928
# of Seqs:    12572575
```

In this example we removed ~27% of all sequences or in other words kept almost 3/4 of our initial fragments for our downstream analysis.

For the next steps close mothur by simply typing *quit()*

> The *out.extendedFrags.qual* file that mothur's fastq.info created can be used in the trim.seqs command to filter out sequences that fall below a certain quality threshold. To include average quality filtering in the trimming step add the name of the quality file with parameter "qfile" and the average quality minimum using parameter "qaverage".
>
> Example:
> Trim as above but also remove all sequences below an average quality score of 25
>
> *mothur > trim.seqs(fasta=out.extendedFrags.fasta,minlength=559,maxlength=561,maxhomop=6,maxambig=0,**qfile=out.extendedFrags.qual,qaverage=25**,processors=8)*

Step 4: Dereplication

Dereplication is the identification of identical sequences which are counted and then removed from the fasta file so that each unique sequence is

present only once in our dataset.

There are three reasons to de-replicate your data set:

1. reduce the size of the data set to reduce computation time and storage space
2. Determine the abundance of each unique sequence. This will be used in the next steps to identify chimeric sequences.
3. Remove singletons, i.e., sequences that are only found once in your data set.

For the next steps of the analysis we will use the software vsearch, which is an open-source implementation of the commonly used *usearch* progr am.

```
$> vsearch --derep_full out.extendedFrags.trim.fasta --output
out.extendedFrags.trim.derep.fasta --minuniquesize 2 --sizeout

  Reading file ./out.extendedFrags.trim.fasta 100%
  7038961814 nt in 12572575 seqs, min 559, max 561, avg 560
  Dereplicating 100%
  Sorting 100%
  10049314 unique sequences, avg cluster 1.3, median 1, max 8654
  Writing output file 100%
  703332 uniques written, 9345982 clusters discarded (93.0%)
```

The command line output of vsearch tells us that it found 10049314 unique sequences with an average abundance of 1.3 and a maximum of 8645. 703332 sequences were kept as unique to determine operational taxonomic units in the downstream analysis which are written to a new file *out.extendedFrags.trim.derep.fasta*. Additionally, vsearch added the qualifier "size" to each sequence in the output file that states how many times the sequence was found in the input file.

Step 5: OTU picking

OTU picking is the process of clustering the trimmed dereplicated sequences into groups of sequences that share a defined sequence similarity and to determine one representative sequence of each group (often referred to as the centroid of the cluster). The idea behind this process is that e.g. clustering 16s rRNA sequences at 97% identity should result in representative sequences that represent the different species present in the given data set.

We will use vsearch for OTU clustering, too.

```
vsearch --cluster_fast out.extendedFrags.trim.derep.fasta -id 0.97
--sizeorder --sizein --sizeout --relabel OTU_ --centroids otus.fas
```

The different parameters of this step mean:

- id 0.97 - this is the identity threshold, i.e. 97% sequence similarity
- sizeorder - orders the different OTUs by abundance determined in the last step
- sizein - simply states that the input file has the size qualifier added (done in the previous step with parameter --sizeout)
- sizeout - as before, add the size qualifier to the OTU for subsequent chimera removal
- relabel - renames the representative/centroid sequences in the output file to "OTU_1, OTU_2 ... OTU_N"
- centroids - the output fasta file name with the representative sequences for each cluster

---

Step 6: Chimera removal

Chimeras are sequences that are a result of the combination of two different sequences into one read. Chimeras originate from amplification/library creation errors where two (similar) sequences are accidentally joined and subsequently amplified.

There are two ways of chimera identification:

1. reference based
2. de-novo

Reference based chimera detection uses known curated sequences, e.g. from the GreenGenes or Silva databases for 16s rRNA genes, to identify OTUs that match partially to one sequence in the database and partially to another one. These sequences are then removed from the OTU file.

De-novo chimera detection uses the OTU set itself as references hereby using more abundant OTUs as references for less abundant ones, assuming that the original/parent sequences are more abundant than the chimeric sequences that originated from them.

For reference based chimera check of your *otus.fas* use

```
vsearch --uchime_ref otus.fas --chimeras otus.chimeras.fas --nonchimeras
otus.clean.fas --db reference.fas --xsize
```

Here *reference.fas* is the fasta file that includes your reference sequences, e.g. silva or green genes 16S rRNA sequences.

For de-novo chimera check of your *otus.fas* use

```
vsearch --uchime_denovo otus.fas --chimeras otus.chimeras.fas
--nonchimeras otus.clean.fas --xsize
```

Both commands will create two output files:

1. *otus.chimeras.fas* which contains the chimeric sequences that were filtered from your data set
2. *otus.clean.fas* which is your final otu fast file.

Step 7: Create final OTU table

In this step we will use our initial non-dereplicated but quality trimmed fragment file from *Step 3* and map all sequences to our OTU representative sequences. This will give us a final OTU table including fragment counts per OTU per sample.

To do this we first have to add a sample qualifier to each of our fragments using the custom script *addSampleQualByFastq.pl*.

```
$>addSampleQualByFastq.pl -q FASTQ_DIRECTORY -f FASTA_FILE -o
OUTPUT_FILE_NAME -m MAPPING_FILE
```

where:

- FASTQ_DIRECTORY is the directory of the fastq files of your initial dataset.
  The script will look for forward read files, i.e., files with *R1* in the name to identify fragments from this sample
- FASTA_FILE is the fasta file you want to add the sample qualifier to, in the case of this tutorial the *out.extendedFrags.trim.fasta*
- OUTPUT_FILE_NAME is the name you want to give the output file, e.g. *out.extendedFrags.trim.added_sample.fasta*
- MAPPING_FILE is, surprise surprise, a simple mapping file that assigns a sampleID to each of the fastq files (Note: these are the demultiplexed fastq files from Step1, i.e., each fastq file contains only reads from one sample!). One line should include a fastq file name and a sampleID separated by a space or tab.

```
S20_13.R1.fastq  FTP109_18
S21_14.R1.fastq  FTP109_37
...
```

Note that you don't have to give the full path to the fastq files just the name.

What did the script do?

Before adding the sample qualifier/attribute to your out.extendedFrags.trim.fasta file it looked like this:

```
$>head out.extendedFrags.trim.fasta
>M02181_102_000000000-D26GT_1_1101_10000_10330
ACTAAAGTTTATTAACACTAAAGGACAATAAAACTTGGGTAACATTCTCAATACAAATATTCTTATT
GCATGCCAGCTAAAGTAGTAAATTATTGGTTTTCTCCTAAGACC
>M02181_102_000000000-D26GT_1_1101_10000_14016
CCGAAGATCAAGCACACTAAAGGACAAAAAGACCCTCTGAAGCTTTATAGGGATAAAACTTTGGCAC
CAAACCTTTATAAAATTGCTATCACCTGCACAAATACCTCTTAAAGAGGCTTGCTTGAGCCTAGGTG
TCAGTTGGGATTTAAGATCAAGACC
>M02181_102_000000000-D26GT_1_1101_10000_15212
ACTAAAGTTTATTAACACTAAAGGACAAAAAGACCCTCTGAAGCTTTATAGGGATAAAACTTGGGTA
ACATTCTCAATACAAATATTCTTATTGCATGCCAGCTAAAGTAGTAAATTATTGGTTTTCTCCTAAG
ACC
...
```

After running the script *addSampleQualByFastq.pl* your out.extendedFrags.trim.added_sample.fasta file looks like this:

```
$>head out.extendedFrags.trim.added_sample.fasta
>M02181_102_000000000-D26GT_1_1101_10000_10330;sample=FTP109_18;
ACTAAAGTTTATTAACACTAAAGGACAATAAAACTTGGGTAACATTCTCAATACAAATATTCTTATT
GCATGCCAGCTAAAGTAGTAAATTATTGGTTTTCTCCTAAGACC
>M02181_102_000000000-D26GT_1_1101_10000_14016;sample=FTP109_37;
CCGAAGATCAAGCACACTAAAGGACAAAAAGACCCTCTGAAGCTTTATAGGGATAAAACTTTGGCAC
CAAACCTTTATAAAATTGCTATCACCTGCACAAATACCTCTTAAAGAGGCTTGCTTGAGCCTAGGTG
TCAGTTGGGATTTAAGATCAAGACC
>M02181_102_000000000-D26GT_1_1101_10000_15212;sample=FTP109_33;
ACTAAAGTTTATTAACACTAAAGGACAAAAAGACCCTCTGAAGCTTTATAGGGATAAAACTTGGGTA
ACATTCTCAATACAAATATTCTTATTGCATGCCAGCTAAAGTAGTAAATTATTGGTTTTCTCCTAAG
ACC
>M02181_102_000000000-D26GT_1_1101_10000_16315;sample=FTP109_18;
ACTAAAGTTTATTAACACTAAAGGACAATAAAACTTGGGTAACATTCTCAATACAAATATTCTTATT
GCATGCCAGCTAAAGTAGTAAATTATTGGTTTTCTCCTAAGACC
```

As you can see each sequence now has a **sample=** added behind its ID which enables vsearch to identify which fragment belongs to which sample.

Note: If the script can't find assign a sample_id for a sequence it will be removes from the output file! For example, in case the fastq files used for joining (Step 2) included reads from other experiments, i.e., the fastq files have not been demultiplexed before joining, the script will remove those reads/fragments from the output file.

Check your file

In this step use the *head* command to see if everything went ok and your resulting fasta file is

1. not empty
2. the fragments have an added *sample* qualifier.

If this step went wrong you might end up with an OTU table with only one sample

Now we can use the new fasta file to create our final OTU table with vsearch. We will again use 97% identity as the similarity cut-off.

```
vsearch --usearch_global out.extendedFrags.added_sample.fas --db
otus.clean.fas --biomout otu_table.biom --otutabout otu_table.tab --id
0.97
```

This will create the final otu_table in two different formats: a biom file that we will use for further analysis and a tab-separated file that can be opened with excel to get a first idea of our data and to check that everything went ok.

Step 8: Taxonomic classification

To assign taxonomies to our OTUs we will use the widely used amplicon data analysis framework qiime. The reason is that qiime offers multiple methods to assign taxonomies to OTU sequences including blast, rdp, uclust and others.

To assign taxonomies to your OTUs you'll need

1. a set of reference sequences in fasta format
2. a mapping file that assigns a taxonomy to each of the reference sequences. An example file might look like this:

```
339039
Bacteria;Proteobacteria;Alphaproteobacteria;Rhodospirillales;unclassifie
d_Rhodospirillales
199390
Bacteria;Chloroflexi;Anaerolineae;Caldilineae;Caldilineales;Caldilineace
a;unclassified_Caldilineacea
370251
Bacteria;Proteobacteria;Gammaproteobacteria;unclassified_Gammaproteobact
eria
11544
Bacteria;Actinobacteria;Actinobacteria;Actinobacteridae;Actinomycetales;
unclassified_Actinomycetales
460067     Unclassified
256904     Bacteria
286896
Bacteria;Actinobacteria;Actinobacteria;Actinobacteridae;Actinomycetales;
Micrococcineae;Micrococcaceae;Kocuria
127471
Bacteria;Bacteroidetes;Sphingobacteria;Sphingobacteriales;Crenotrichacea
e;Terrimonas
155634
Archaea;Euryarchaeota;Methanobacteria;Methanobacteriales;Methanobacteria
ceae;Methanosphaera
```

where each line contain a sequence identifier (in this case numbers) and a string of taxonomic levels separated by ';'.

Let's assign taxonomies to our OTUs. As an example we will not use the default method *uclust* but specify *blast* as the method of choice in qiimes *assign_taxonomy.py* script:

```
# use qiime scripts
assign_taxonomy.py -i otus.clean.fas -r reference.fas -t
reference_taxonomy.tab -m blast
```

Silva

Reference and taxonomy files for Silva 16S and 18S sequences can be found on here

This will create a folder called *blast_assigned_taxonomy* in your current working directory which include two files:

1. otus.clean_tax_assignment.log
   This contains useful information, e.g. how many of your OTUs could not be identified using your reference
2. otus.clean_tax_assignment.txt
   This is the actual taxonomy file that shows the taxonomy of every OTU, the e-value of the reference blast hit and the reference sequence id

```
OTU_321
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_C;Subclade_C;C; 2e-63   50300444
OTU_320
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F;F; 2e-45   572065117
OTU_323
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F;F; 6e-36   572065117
OTU_484
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F3;F3.2  4e-30   408777588
OTU_485
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_B;Subclade_B;B; 2e-23   18026233
OTU_487
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F3;F3.2  4e-30   408777588
OTU_480
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_C;Subclade_C;C; 2e-32   296245017
OTU_481
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F;F; 7e-29   83626842
OTU_482
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_C;Subclade_C;C; 2e-32   296245017
OTU_483
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F4;F4.1  2e-32   408777589
OTU_248
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_F;Subclade_F3;F3.2  7e-54   408777588
OTU_247
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade
_A;Subclade_A2;A2    2e-72   344227659
```

Before we can now use this taxonomy we will have to add a header to it so that the next program knows which column contains the taxonomy.

To do this open the file in your preferred text editor and add the following header line as a comment:

```
#OTUID taxonomy evalue  taxonid
OTU_321
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_C;Su
bclade_C;C; 2e-63   50300444
OTU_320
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_F;Su
bclade_F;F; 2e-45   572065117
OTU_323
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_F;Su
bclade_F;F; 6e-36   572065117
OTU_484
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_F;Su
bclade_F3;F3.2  4e-30   408777588
OTU_485
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_B;Su
bclade_B;B; 2e-23   18026233
OTU_487
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_F;Su
bclade_F3;F3.2  4e-30   408777588
OTU_480
Eukaryota;Dinophyceae;Suessiales;Symbiodiniaceae;Symbiodinium;Clade_C;Su
bclade_C;C; 2e-32   296245017
```

In case you don't know how to use a text editor on Unix systems ("the command line") I recommend that you do a short tutorial on either vi, nano or emacs which are the most widely used editors on unix systems.

This will not only be useful for this tutorial but for almost everything you do on the command line. Without an editor you will not be able to open or edit any files!

The last thing left now is to add the taxonomies of each OTU to our OTU table using our edited taxonomy file and the *biom* package:

```
biom add-metadata -i final_otu_table.biom -o
final_otu_table.added_taxonomy.biom --observation-metadata-fp
otus.clean_tax_assignments.txt --sc-separated taxonomy
```

Congratulations, your final OTU table is ready for analysis!