V6-V8 sequence processing pipeline

September 7, 2020

because, well, need a document to pick things up and put them down....

Contents

1 MVCO sample and sequencing info

1.1 Sampling

Sampling for DNA at MVCO started late 2010 and has been ongoing since. Throughout the time series, sampling and filter methods have changed. For the majority of samples, surface seawater was filtered onto a 0.22 μm Sterivex filter via vacuum (with suction no more than 5-10 mmHg). This process, depending on the sample, would usually take an hour (but sometimes could be much longer). Bottom samples were also taken in the early years (to the best of my knowledge, these remain un-extracted in the freezer). Around fall 2017, with many cruises on the horizon, we realized that a peristaltic pump would speed up the filtering dramatically (and is often used in other labs). We weren't sure though of the differences downstream, so duplicate samples were taken and filtered both ways; peristaltic and vacuum for later comparison. Then with the LTER change over, coupled to the cost of Sterivex filters, time series samples starting early 2018 were filtered onto PES disk membranes via vacuum filtration. Again, duplicate samples were taken and processed both ways. The follow table lists how different samples were filtered. Mock community samples were also sequenced. These consisted of different types of *Synechococcus* to better understand process bias for downstream *Synechococccus* sequence compositions.

Filter	Method	MVCO Samples	Mock/test samples
Sterivex	Vacuum	254-387, 400	MVM1, MVM2
Sterivex	Peristaltic	388-393	SynCommunityA-C
PES disk filter	Vacuum	393-399, 400	SynCommunityD

Table 1: Filter types and pump used for each sample.

Other samples from an OOI cruise late October (AR24B) were also sequenced along with

1.2 V6V8 Primers and Sequencing runs

The primers generally found in the literature for the V6-V8 region are:

- 926F: 5'- AAACTYAAAKGAATTGACGG-3'
- 1392R: 5'-ACGGGCGGTGTGTRC-3'

Combined with the Illumina adapters and sequencing primers, the final primers used in the reactions were:

barcode	seq	full forward primer
bc1	CGTCT	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTCGTCTAAACTYAAAKGAATTGACGG
bc2	CTCGT	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTCTCGTAAACTYAAAKGAATTGACGG
bc3	CTCTA	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTCTCTAAAACTYAAAKGAATTGACGG
bc4	GACGT	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTGACGTAAACTYAAAKGAATTGACGG
bc5	GATCT	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTGATCTAAACTYAAAKGAATTGACGG
bc6	ACACT	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTACACTAAACTYAAAKGAATTGACGG
bc7	AGCTA	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTAGCTAAAACTYAAAKGAATTGACGG
bc8	TCGTC	AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCTTCGTCAAACTYAAAKGAATTGACGG
index	seq	full reverse primer
idx118	GTATTT	CAAGCAGAAGACGGCATACGAGATGTATTTGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx119	TATCTA	CAAGCAGAAGACGGCATACGAGATTATCTAGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx120	TTGCCA	CAAGCAGAAGACGGCATACGAGATTTGCCAGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx121	AGTAGC	CAAGCAGAAGACGGCATACGAGATAGTAGCGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx122	GCAATT	CAAGCAGAAGACGGCATACGAGATGCAATTGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx123	CATACC	CAAGCAGAAGACGGCATACGAGATCATACCGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx124	ATGTGT	CAAGCAGAAGACGGCATACGAGATATGTGTGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC
idx125	CCTGCG	CAAGCAGAAGACGGCATACGAGATCCTGCGGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCTACGGGCGGTGTGTRC

Table 2: Forward and reverse primers used. Forward primer has 926F portion in red, barcode in blue), reverse primer has 1392R in red, index in blue, where R = A or G, Y = C or T, and K = G or T.

Libraries were prepped according to Keck Sequencing Facility protocols, as described for V4V5 region at https://vamps.mbl.edu/resources/primers.php. Below lists the 6(7) sequencing runs over which the MVCO timeseries and mock community samples were sequenced:

Run	Date	Sample list
1	Oct-31-2016	294,302,326,363A
2	Jan-4-2017	254,257,259,262,265,267,271,273,275,278,280,282,287,289,294,296,302,304,306, 308,310,312,314,316,318,320,322,325,327,329,331,333,336,338,340,342,344,347, 349,351,353,355,357,359,361,363A,364,366,368,371,373,375, M2
3	Feb-6-2017	258,260,261,263,266,270,272,274,276,279,281,286,288,290,295,300,303,305,307, 309,311,313,315,317,319,321,323,324,326,328,330,332,335,337,339,341,343,348, 350,352,354,356, 358,360,362,363B,365,367,369,370,372,374, M1
4	Aug 4 2018	260,294,302,317,326,363A
5	Nov 12 2018	263,325,336,337,339,357,358,376,377,378,379,380,381,382,383,384,385,386, 387,388,389,390,391,392,393
6	Feb 21 2019	308,309,310,338,394,395,396,397,398,399, 400Milli, 400Ster, SynCommunityA, SynCommunityB, SynCommunityC, SynCommunityD
7	Jan 13 2020	308,309,310,338,394,395,396,397,398,399, 400Milli, 400Ster, SynCommunityA, SynCommunityB, SynCommunityC, SynCommunityD

Table 3: Date and samples sequenced over 7 different sequencing runs. Note that run 6 did not go well (read quality drops off very sharply and merging was not really possible), and was re-sequenced from the same library for run 7.

2 Data processing and quality control

2.1 Demultiplexing

Once the sequencing run is complete, reads are partially demulitplexed (separated into samples) by index on the machine where the raw image files were written during the run. Each set of index reads contains multiple barcodes and those can be demultiplexed a number of ways. The script `demulitplex.sh` accomplishes this given a file that lists the samples, barcodes and indexes used. This script should be run inside the directory with the raw files. The products are sample-named fastq files plus a `demulitplex_logfile.txt` that shows how many reads separated into the samples, and reads for which no barcode could be found. Alternatively, the script `iu-demultiplex` from illumina-utils (mentioned below) will also accomplish this given an input file of indexes and barcodes.

2.2 FASTQC

After demultiplexing the reads, it's a good idea to get a handle on the quality. This is accomplished with the program FastQC (see installation instructions in section 6). A bash script (`fastqc.sh`) calls this program via command line to run FastQC on all the reads. This first part of the script is as follows:

```
samplelist=$(ls | grep -oP "[A-Z,0-9,\-,a-z]*(?=\_R1)")

if [ ! -d "fastqc_output" ]
then
echo "making output directory"
mkdir fastqc_output
fi

for sample in $samplelist
do
echo "fastqc ${sample}"
fastqc ${sample}_R1.fastq -extract -o fastqc_output
fastqc ${sample}_R2.fastq -extract -o fastqc_output
done
```

FastQC has great output and diagnostics. In particular, the per base sequence quality is really informative and very important to get a handle on for merging down stream. The remainder of `fastqc.sh` creates a short summary table to see per sample at which position quality dips below a score of 30.

2.3 Merging

The read merging and filtering is done with illumina-utils from Meren (see Section 6 for some details about installation, but original site has great tutorials and installation notes). The script `run_merge.sh` first set ups required initialization file (`*.ini`) necessary for the merging script, `iu-merge-pairs` to run. Currently, `illumina-utils` are in python 3, which is not the normal default on my machines, such that they are called with a virtual environment. Sections of the `run_merge.sh` are shown and explained below:

The first section sets up the .ini files needed to merge. Alternatively, this could be achieved with **iu-gen-configs** that would generate the `.ini` config files if the forward and reverse fastq files have already been made and you have a tab-delimited file containing the names such as:

```
sample        r1        r2
MV254        MV254.R1        MV254.R2
MV302        MV302.R1        MV302.R2
(...)        (...)        (...)
```

However, I found it easier to just make them in the same script as follows. First, Python fancy magic to run:

```
source ~/virtual-envs/illumina-utils-v2.0.2/bin/activate
```

```
filelist=$(ls | grep -oP "[A-Z,0-9,\-,a-z]*(?=\_R1|\.R1)")

for f in $filelist
do
echo $f
echo "[general]" >> ${f}.ini
echo "project_name = ${f}" >>  ${f}.ini
echo "researcher_email = khunter-cevera@mbl.edu" >> ${f}.ini
echo "input_directory = /path/to/demulitplexed_reads" >> ${f}.ini
echo "output_directory = /path/to/demulitplexed_reads/merged_reads" >> ${f}.ini
echo "[files]" >> ${f}.ini
echo "pair_1 = ${f}_R1.fastq" >> ${f}.ini
echo "pair_2 = ${f}_R2.fastq" >> ${f}.ini

# NO LONGER USING THIS OPTION:
# echo "[prefixes]" >> ${f}.ini
# echo "pair_1_prefix = .....AAACT[CT]AAA[GT]GAATTGACGG" >> ${f}.ini
# echo "pair_2_prefix = ACGGGCGGTGTGT[AG]C" >> ${f}.ini
done

mv *.ini merged_reads/
cd merged_reads
```

This creates a configuration file like this for each sample:

```
[general]
project_name = MVXXX
researcher_email = khunter-cevera@mbl.edu
input_directory = ./
output_directory = ./

[files]
pair_1 = MVXXX.R1
pair_2 = MVXXX.R2
```

Optional is the inclusion of primers and/or barcodes to be removed. These would be listed in the prefixes section commented out in the above code:

```
[prefixes]
pair_1_prefix = ^.....AAACT[CT]AAA[GT]GAATTGACGG
pair_2_prefix = ^ACGGGCGGTGTGT[AG]C
```

Primers/prefixes can be specified with regular expressions, and if prefixes are defined, the results will only contain pairs that match them. The illumina-utils scripts exclude any read without a match. However, I often found that high quality sequences can be contained within a primer set that is just one or two bases off of the expected primers (with the regular expressions as described above). So that section is no longer included in the .ini files that we construct to avoid throwing away good data. Primers are dealt with after the merging (see section below).

The next section of the script performs the merging by calling `iu-merge-paris`:

```
filelist=$(ls *.ini)

N=6 #refers to number of cores on your machine
(
for sample in $filelist
do
((i=i%N)); ((i++==0)) && wait
echo $i
iu-merge-pairs --max-num-mismatches 3 --enforce-Q30-check ${sample} &
done
)
```

These are clever bash commands (thank you stackoverflow) that allow for batch processing of each file in chunks of 6. The 'wait' command pauses the for loop until all 6 processes have finished running, while the '&' causes each process to run in the background on a separate thread. The output files are simple fasta format (no quality scores) in lower case letters, but mismatched bases in the overlap region are shown with a capital letter (chosen based on highest quality score between the two reads). Additional information is in the header, such as number of mismatches, number of bases passing Q30 check, etc (see Meren's site for full info).

And then close fancy python environment with:

```
deactivate
```

If additional filtering is required (e.g. only keep reads with perfect merge), this can be done with an additional call to **iu-filter-merged-reads**, which can be used in the following within a for-loop:

```
for filenum in `cat list`; do iu-filter-merged-reads -m 0 ./${filenum}_MERGED; done
```

where 'list' is the list of dataset names. The -m flag means no more than 0 mismatches allowed.

To remove the primers on the merged reads, the script `primer_wrapper.sh` is called that uses a short python script (`remove_forward_reverse_primers.py`) to remove primers, but critically, allows up to 2 or 3 mismatches, insertions or deletions for primer detection:

```
samplelist=$(ls | grep "MERGED")

N=6 #refers to number of cores on your machine
(
for sample in $samplelist
do
((i=i%N)); ((i++==0)) && wait
echo " ${sample}"
python ~/Documents/V6V8_analysis/scripts/raw_seq_processing_QC_scripts/
↪  remove_forward_reverse_primers.py ${sample} &
done
)
```

2.4 Summary of data processing and merging scripts

Script	Description	Inputs	Outputs
`demulitplex.sh`	Separates sample reads based on index and barcodes	list of barcodes and indexes for each sample	demulitplexed reads
`fastqc.sh`	Calls FastQC for each forward and reverse sets of reads	demultiplexed reads	FastQC output plus summary tables
`run_merge.sh`	Calls `illumina-utils` to perform merging, creates necessary .ini files	demultiplexed reads	merged reads, log files, .ini files
`merge_tally.sh`	Reads STATS files generated from `illumina-utils` and creates table of output for all samples	merged output stats	summary table
`primer_wrapper.sh`	Calls `remove_forward_reverse_primers.py` to remove primers	merged reads	reads with primers removed
`remove_forward_reverse_primers.py`	script that allows for primer identification of up to 3 mismatches, insertions or deletions	sequences with primers	sequences without primers

Table 4: Scripts listed can be found at NES_LTER/amplicon_sequencing/V6V8/scripts/raw_seq_processing_QC_scripts/

3 Taxonomy Assignment

3.1 VAMPs assignment

Taxonomy can be assigned using VAMPS (vamps2.mbl.edu). Data can be uploaded as a multi-dataset file, which can be created by simply concatenating all the merged (primers removed) reads into a giant file (which should be gzipped for maneuverability). The script `vamps_upload_seqs.sh` organizes sequences based on projects (the 6 runs contain MVCO time series samples, OOI samples and mock community samples), and changes the sequence identifier so that is formatted for VAMPS (needs to have a space after unique ID; does not accept any '|').

Once uploaded to VAMPs, I used GAST to assign taxonomy (could also use RDP). Once finished, the results can be exported as data tables with taxonomic identification. I usually only export the following two products (under 'data exports' and then 'data file retrieval') at species level of all domains (including unknowns):

taxbyseq.tsv -Tab-deliminated columns of counts for each sample for each unique sequences with taxonomy of that sequence; column headers appear as:

- seqrefid, Sample 1, Sample 2,....,Sample n, Distance, Sequence , Rank, and Taxonomy

fasta file (.fa) -unique sequences per sample (such that some sequences may be repeated in this file); number of times sequence was observed per sample can be found in the header with optional taxonomy:

- >137642_0|MVCO_V6V8_run1|MV254|0.2%|1
- >1326981077|MVCO_timeseries_run7–308|0.12600|Archaea;Euryarchaeota;Halobacteria; Halobacteriales |frequency:1

A very useful check throughout the entire pipeline is simply counting the number of sequences before and after and making sure they are all accounted for! For example, to compare the number of sequence counts uploaded to VAMPs and the number of counts retrieved:

Number of uploaded sequences:

```
grep -c ">" seq.fasta
```

A bit of a fancier awk script (thanks stackoverflow) to sum column counts in tax-by-seq table:
First, find the number of columns:

```
awk 'NR>1{print NF; exit}' mvco_timeseries_taxbyseq.tsv
```

'NR¿1' tells awk to skip the first line, which has other stuff on it from the tax-by-seq table. The last four columns contain sequence and taxonomy info, which we don't want to sum over, so exclude those:

```
awk '{for (i=1;i<=39;i++) sum[i]+=$i;}; END{for (i in sum) print sum[i];}'
```

The result are column sums, and this could just be piped back to awk again to sum further (lazy, and there much better ways to write this, but it works :)

```
| awk '{s+=$0}{print s}'
```

The ending numbers of each should match!

3.1.1 Other extractions from VAMPS tables and outputs

Using *Synechococcus* as an example, one can extract information about this species directly from the tax-by-seq table with different bash commands. A few examples are below and have been useful in the past.
Extract lines that refer only to "Synechococcus" sequences:

```
sed -n 2p taxbyseq.tsv > synonly_table.txt
grep "Synecho" taxbyseq.tsv >> synonly_table.txt
```

The first command copies over the column headers, the second finds all lines that contain "Synecho" (listed in the taxonomy).

To pull out only certain samples from a taxbyseq table, first, find the column number(s) of the desired sample(s). In the below example, this is 'MVM1' or 'MVM2':

```
sed -n -e 2p taxbyseq.tsv | tr '\t' '\n' | grep -n 'MVM'
```

The -n flag with grep returns a line number, which is used in the following to create a condensed table which has sequence and counts for two samples, but only for those which are not zero:

```
cut -f54,56,57,58 taxbyseq.tsv | awk ' { if ($1 != 0) { print } } ' >
↪  mock_culture2_taxbyseq.txt
```

If you need to parse the fasta file by sample, and collect corresponding sequences, this can be achieved with grep. For example:

```
grep -A1 -E --no-group-separator "MVM[0-9]{1}" fasta-mvco-2010_2016.taxa.fasta >
↪  mock_seqs.fasta
grep -A1 -E --no-group-separator "MV[0-9]{3}" fasta-mvco-2010_2016.taxa.fasta >
↪  mvco_env_seqs.fasta
grep -A1 -E --no-group-separator "\-\-[0-9]{3}" fasta-nov2018.taxa.fasta >>
↪  mvco_env_seqs.fasta
grep -A1 -E --no-group-separator "C[0-9]{1,2}N[0-9]{1,2}|BUCKET"
↪  fasta-nov2018.taxa.fasta >> OOI_seqs.fasta
```

The -A1 flag prints the line following the match, -E tells grep to use regular expressions, and –no-group-separator prevents empty space after the matches.

The fasta file from VAMPS only contains unique sequences. To see how many total sequences there were, this can be tallied from the frequency count that appears at end of the sequence ID header:

```
awk -F: 'NR%2{SUM+=$2}END{print SUM} mock_seqs.fasta'
```

The '-F' indicates that the field separator will be ':', the NR%2 means go every other line for the tally that is recorded in 'SUM', and then print this value at the end.

3.2 BLAST assignment

As a backup, we checked assignments for *Synechococcus* designation by blasting against a custom database of full length *Synechococcus* sequences. Mainly, this was just for sanity check (although in the past, I've come across bugs in either the GAST assignment or VAMPs pipeline that doesn't always provide the correct assignment or numbers).

3.2.1 *Synechococcus* database construction

Based on publications and searches for *Synechococcus* 16S rRNA sequences in Genbank, we were able to download 232, 39 of which are from MVCO isolates. The following publications were used as the primary sources for the majority of 16S sequences and for clade designations:

- Ahlgren & Rocap 2012
- Choi & Noh 2009
- Everroad & Wood 2012
- Fuller et al. 2003 (exclusive of Ahlgren & Rocap 2012)
- Mazard et al. 2012 and other sequences not listed in SI excel spreadsheet

Searching in Genbank yielded additional sequences, mostly from whole genome shotgun sequences, but often clade designation could be not be determined. Any sequence for which clade designation was ambiguous was not used. The script that does some of this sorting of the database is `update_marine_syn_database.jl`. The sequences in our databse is certainly not exhaustive, but a good start :) These sequences are formed into a searchable blast database as follows:

```
makeblastdb -in
↪  ~/V6V8_analysis/Marine_Syn_16S_database/full_length_16S_TOTAL_seqs.fasta -dbtype
↪  nucl -out syn_16S_db -title  "Full-length marine Synechococcus 16S sequences"
```

Blast is called in a wrapper script (`run_blast.sh`) to search against this custom *Synechococcus* database and provide output results. The call to `run_blast.sh` is further embedded in a julia script that handles most of this processing. The scripts `mock_community_blast_screen_and_oligotype_format_out.jl` and `mvco_timeseries_blast_screen_oligotype_format_out.jl` import the tax-by-seq tables downloaded from VAMPs, create unique sequence identifiers for the sequences, exports to blast, re-imports the results, sorts and exports the total *Synechococcus* sequences in a format necessary for oligotyping (see below).

In general, the blast check didn't reveal too many sequences that needed to be added (and none removed) from the *Synechococcus* GAST pool.

3.3 Oligotyping

While we have taxonomy classifications, oligotyping is useful to 1) define diversity below the species level and 2) reduce sequence noise (via clustering).

First, sequences must be formatted for oligotyping: this includes changing sequence IDs and replicating sequences the number of times that they appear in a sample. (Recall that the tables or fasta files from vamps return only unique sequences). This has already been accomplished in the previous

scripts.

Then we must align the *Synechococcus* sequences before we can perform oligotyping. While this might not be needed for Illumina type sequencing (as errors are usually base changes rather than insertions and deletions), it is worth aligning, just in case. This is done with the PyNAST aligner against either a *Synechococcus* reference alignment or a against a larger, more curated alignment from the Green genes database, which can be found at http://greengenes.lbl.gov/Download/OTUs/gg_otus_6oct2010/rep_set/gg_97_otus_6oct2010_aligned.fasta. This can be downloaded by:

```
curl "[green genes website adress]" > path/to/gg_97_otus_6oct2010_aligned.fasta
```

We then align the *Synechococcus* sequence reads:

```
pynast -i synseqs_for_oligotyping.fasta
-t /path/to/gg_97_otus_6oct2010_aligned.fasta -l 300
```

The '-l' flag is important; it sets the minimum sequence length to be considered for an alignment, and the default is set at 1000 base pairs. The alignment comes with a lot of uninformative gaps (from alignment with other bacteria in the reference file), and these can be removed with a command from within the oliogtyping package:

```
o-trim-uninformative-columns-from-alignment
 synseqs_for_oligotyping_pynast_aligned.fasta
```

This results in a '-TRIMMED' file for use. Before we start oligotyping, we can check that the file is okay, by running the oligotyping command:

```
o-get-sample-info-from-fasta mvco_env_only_synseqs_allruns_for_oligotyping.fasta-TRIMMED
```

Now, we are ready to start the oligotyping. This first step is to generate an entropy file for each nucleotide position with the command:

```
entropy-analysis synseqs_for_oligotyping_pynast_aligned.fasta-TRIMMED
```

This file is used to do the oligotyping analysis. This process can be quite iterative and Meren has some nice blog posts about how to perform oligotyping responsibly (http://merenlab.org/2013/11/04/oligotyping-best-practices/). A good place to start is the entropy diagram and see which positions have the highest entropy. One chooses the positions that have entropy or all those that are above an entropy threshold (the parameter c tells the program to chose the X number of positions that have the highest entropy, while C allows the user to specify those exact positions). Some other nice metrics are the 'minimum substantive abundance', which is the minimum number of times that the most common sequence for an oligotype must occur (-M) and the total min abundance of an oliogtype (-A) to be kept as an oligotype (over the whole dataset). To run the oligotyping, the following syntax is used, where N1, N2, N3 are integers:

```
oligotype fasta_file entropy_file -c N1 -M N2 -A N3
```

The following was chosen as parameters for the environmental *Synechococcus* :

```
oligotype synseqsfor_oligotyping_pynast_aligned.fasta-TRIMMED
↪  synseqs_allruns_for_oligotyping_pynast_aligned.fasta-TRIMMED-ENTROPY -c19 -M200
```

Once finished, the terminal will give the location of the files for html viewing and you can copy this address into a browser to view the very nice and detailed analysis metrics that the oligotyping scripts provide. If you forget this address, just double click on the 'index.html' in the `HTML-OUTPUT` folder within the oligotyping folder that has been created. The resulting oligotypes are actually a composite of sequences and the most abundant sequence is termed the representative sequence. These are used going forward to assign clade identity based on reference to the 16S *Synechococcus* clade database. The resulting representative oligotype sequences are stored in `OLGIO-REPRESENTATIVES.fasta`, but still have alignment gaps and clunky names. To remove gaps and rename based on abundance:

```
sed 's/[-]//g' OLIGO-REPRESENTATIVES.fasta | awk 'NR % 2 {x=">O"++i; print x} !(NR % 2)
↪  {print}'  > oligo_repseqs.fasta
```

For the *Synechococcus* , it is critical that the oligotypes be linked to a clade ID (as physiology and biogeography has only been described in clades). This is typically done with a more high-resolution diversity marker (i.e. *ntcA*, *petB*, ITS, *rpoC1*) than the 16S (and worse here, a subset of the 16S!). However, we can still try to link these oligotypes to a clade through a database of *Synechococcus* 16S sequences for which clade ID has been assigned with a higher resolution marker. This is done with the script `unique_seqs_marine_syn_db_oligotype_match.jl`.

3.4 Summary of taxa-processing scripts and programs called:

Script/Program	Description	Inputs	Outputs
vamps_upload_seqs.sh	Sort and modify sequence ID headers for upload to VAMPs	merged and primer trimmed fasta files	concatenated fasta file for upload
Prep *Synechococcus* sequences:			
run_blast.jl	Wrapper script to blast	searchable database, fasta file	blast results
mock_community_blast_screen_and_oligotype_format_out.jl	Parse taxbyseq table, run blast, coordinate results and produce fasta file of *Synechococcus* sequences for oligotyping	taxbyseq.tsv	fasta file for oligotyping
mvco_timeseries_blast_screen_oligotype_format_out.jl	Parse taxbyseq table to create fasta file of *Synechococcus* sequences for oligotyping	taxbyseq.tsv	fasta file for oligotyping
Perform oliogytping:			
pynast	perform alignment of template to reference database	sequences to align, reference alignment	aligned sequences, log files
o-trim-uninformative-columns-from-alignment	removes uninformative gaps from alignment (i.e. where a position is a gap for every sequence)	fasta file	-TRIMMED fasta file
o-get-sample-info-from-fasta	make sure file is appropriate for oliogtyping	fasta file	screen output description of samples
entropy-analysis	calculate entropy for each nucleotide position	aligned fasta file	entropy analyis
oligotype	perform oligotyping	aligned fasta file, entropy analysis	folders of oligotyping goodnees
Synechococcus database and clade ID			
update_marine_syn_database.jl	remove sequences that are do not have clear clade ID	fasta file of *Synechococcus* sequences, previous *Synechococcus* databases	database table with sequences
unique_seqs_marine_syn_db_oligotype_match.jl	find unique V6-V8 regions across *Synechococcus* database sequences, assign clade ID to oligotype reference sequence	reference *Synechococcus* database, oligotype sequences	oligotype IDs, heatmap figure table info

Table 5: Scripts listed can be found at NES_LTER/amplicon_sequencing/V6V8/scripts/taxa_processing_scripts/

4 Compositional Data Analysis

We analyzed our sequence data as compositions: please see `github./com/khuntercevera/coda_utilities/` for scripts used and examples.

5 Some Package Installation Instructions

Below are my personal notes for installation of certain packages, mainly on Ubuntu 18.04 (but sometimes on a Mac OS). These notes served for my reference and benefit and are listed below not with any guarantee that they will work exactly in your case, but rather that they may provide some guidance and helpfulness if needed.

5.1 Compiling this tex document with minted package

This document uses minted to nicely define and highlight code blocks. On a mac and with Ubuntu, I had some difficulties getting this package to work. The minted package calls a python package called Pygments (pip install Pygments), however, the Tex paths on mac for some reason or another did not follow the native machine paths where Python is listed (i.e. the shell variable PATH is different than what Tex actually checks). A work around was to make a symbolic link to pygmentize to where Tex actually checks for files via:

```
ln -s /opt/local/Library/Frameworks/Python.framework/Versions/2.7/bin/pygmentize
↪  /usr/local/bin/pygmentize
```

Second, if you've tried to compile with minted and it failed, other compilations for some reason now won't work. Delele the '_minted-XXXX' folder that appears in the tex document folder and things should work.

On Ubuntu, I'm working with TexStudio and needed to run minted with the shell-escape flag set. Following a very nice stackoverflow post, you can make a new command to do just that. Under Options, Configure TexStudio, Build, add a new command that reads:

```
pdflatex -synctex=1 -interaction=nonstopmode --shell-escape %.tex
```

This command should show up under Tools, Users. Execute this to compile!

5.2 FastQC

FastQC runs with java; to install on Ubuntu, first check if installed:

```
java --version
```

If no, then:

```
sudo apt update
sudo apt install default-jre
java --version
```

And then you should see something like this:

```
openjdk 11.0.4 2019-07-16
OpenJDK Runtime Environment (build 11.0.4+11-post-Ubuntu-1ubuntu218.04.3)
OpenJDK 64-Bit Server VM (build 11.0.4+11-post-Ubuntu-1ubuntu218.04.3, mixed mode,
↪  sharing)
```

Download FastQC from https://www.bioinformatics.babraham.ac.uk/projects/download.html#fastqc. I tend to place all of my software in a ~/software directory. To make FastQC operational, I just followed the instructions to make the fastqc wrapper script executable and then callable from anywhere with a symbolic link:

```
chmod 755 fastqc
sudo ln -s ~/software/FastQC/fastqc /usr/local/bin/fastqc
```

Done and ready to go!

5.3 illumina-utils within a virtual Python3 env

We performed read merging with the package illumina-utils, which is written in Python3. This is not the default on my machine. A work around is to have this used with a virtual environment. The command sequence was as follows:

Make sure pip3 is installed, by first checking:

```
pip3 -V
```

If not, then this command did the trick for me:

```
sudo apt install python3-pip
python3 -m pip install --user virtualenv
```

Following Meren's instructions from https://github.com/merenlab/illumina-utils#installing:

```
mkdir -p ~/virtual-envs/
virtualenv ~/virtual-envs/illumina-utils-v2.0.2
source ~/virtual-envs/illumina-utils-v2.0.2/bin/activate
python --version
pip install illumina-utils
```

Check that the output starts with a '3' from calling the python version. On Ubuntu 18.04, I was missing configparser, but this was remedied with:

```
pip install configparser
```

And then enter:

```
deactivate
```

to exit and return to python2 :)

5.4 Blast+

Blast exectuables can be downloaded from ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST/. For ubuntu 16.04 and 18.04, I downloaded ncbi-blast-2.7.1+-x64-linux.tar.gz. This is then extracted and saved in a software folder. I then just needed to add this location to the PATH variable:

```
export PATH=$PATH:$HOME/software/ncbi-blast-2.7.1+/bin
echo $PATH
```

5.5 Oligotyping

Installation also should be just as easy as:

```
pip install oligotyping
sudo apt-get install python-tk
```

although recently I had to do:

```
pip3 install oligotyping
sudo apt-get install python3-tk
```

to get it to work...

5.6 PyNAST

The reference page for PyNAST installation is at http://biocore.github.io/pynast/install.html, and this program can be installed with pip with command:

```
sudo pip install pynast
```

While this worked well on a Mac, for Ubuntu, this appears to be:

```
sudo apt install pynast
```

Before this step though, we need to install the required dependencies. The only one I needed was uclust. It seems that only binaries are available from http://www.drive5.com/uclust/downloads1_2_22q.html. Initially, I was unsure what this file was, but this can be checked with:

```
file uclustq1.2.22_i86darwin64
uclustq1.2.22_i86darwin64: Mach-O 64-bit executable x86_64
```

An executable! On both Mac and Ubuntu, I needed to actually make the file executable with:

```
chmod u+x uclustq1.2.22_i86darwin64
```

PyNAST does not recognize the name 'uclustq1.2.22_i86darwin64' (or other binary names), so to solve, I used a symbolic link and placed the link to 'uclust' in a location listed in my $PATH variable. The original executable was either in Applications or in the ~\software folder:

```
ln -s /path/to/uclustq1.2.22_i86darwin64 /usr/local/bin/uclust
```

There may be other ways around this - directly renaming and placing in the right location, but this way felt safer.