Processing 16S data: an informal primer about 16S rRNA amplicon data

Scott W. Olesen

March 2021

1 Foreword

1.1 Where this document came from

I wrote the first edition of this document as a graduate student for use by my research group, the Alm Lab at MIT. In 2018, I published the second edition online. Now, in 2021, I'm updating this document with a third edition in order to reorganize the material and to include two important new developments: denoising and QIIME 2. I'm grateful to OpenBiome, which gave me the opportunity to do this revision as part of a microbiome science seminar course for Aga Khan University.

1.2 What this document is for

My goal for this document is to help you understand the theory behind 16S data processing. Processing 16S data involves a lot of very small decisions (that probably have a small effect on your results) and a few big decisions (that certainly have a big effect on your results). The nice pipelines make it possible for you to shut your eyes to these complications. In contrast, I want to empower you to be able to critique and doubt other people's methods.

1.3 What this document is not

This document is not perfect. It is full of my own ignorance, ideas, and opinions. Take it with a grain of salt.

This document is not a literature review. Next-generation sequencing for microbial ecology is a large field, and here I just scratch the surface.

This document is not a tutorial. If you desperately need to turn some raw data into OTU tables in the next 10 hours, don't read this document. If you want to know something more principled about *how* to turn fastq's into OTU tables, read on.

2 Where 16S data comes from

2.1 What is 16S data?

By “16S data” I mean metagenomic amplicon sequencing of some section of the bacterial 16S rRNA gene.

The use of the term “metagenomic” in this context may be confusing, since “metagenomics” is often used interchangeably with “shotgun whole-genome metagenomic sequencing”. Technically, “metagenomic” means “related to more than one genome”, that is, sampling from an entire community rather than from a single cell or a single colony. Both 16S sequencing and whole-genome shotgun sequencing intend to sample from many bacterial species in the sample, so they are both technically metagenomic.

It is also worth noting that, while amplicon sequencing in theory provides strictly less information that shotgun sequencing —since it examines only a part of bacteria’s genomes rather the entire genomes— amplicon-based approaches have three key advantages.

First, shotgun sequencing projects tend to be more expensive, since they need to sequence deeper to get the same information about microbial community composition. In theory, if you shotgun sequence at a great enough sequencing depth, you could reconstruct all the information that you could get from 16S amplicon sequencing, but this can be an expensive proposition.

Second, because only bacteria and archaea have the 16S gene,[1] a tube of 16S amplicon DNA mostly carries information about microbes. In contrast, the majority of shotgun reads from, say, a swab of human skin will be human DNA. Amplicon sequencing may be some help in terms of privacy, since research subjects are likely more comfortable with their microbes’ DNA being sequenced rather than their own human DNA. Furthermore, if you are interested in only microbes, using amplicon sequencing means that you are not spending any of your DNA sequencing budget on sequencing human DNA that is unimportant to your research question.

Finally, amplicon sequence data is substantially easier to work with from a bioinformatic point of view. Shotgun sequences need to be *assembled* to recreate the genome sequence, which is computationally and conceptually demanding. By contrast, 16S sequencing data provides information from just one part of the genome, and each read is likely to cover the entire area of interest.

[1] Chloroplasts, found in algae and other eukaryotes, have a ribosomal gene that is very similar to 16S and often ends up getting amplified in 16S data sets.

2.2 The 16S gene

All bacteria and archaea have at least one copy of the 16S gene in their genome.[2] The gene has some sections that are *conserved*, meaning that they are very similar across all bacteria, and some sections that are *variable* (or "hypervariable"). The idea behind 16S sequencing is that the variable regions are not under strong evolutionary pressure, so random mutations accumulate there. Closely-related bacteria will have more similar variable regions than distantly-related bacteria.

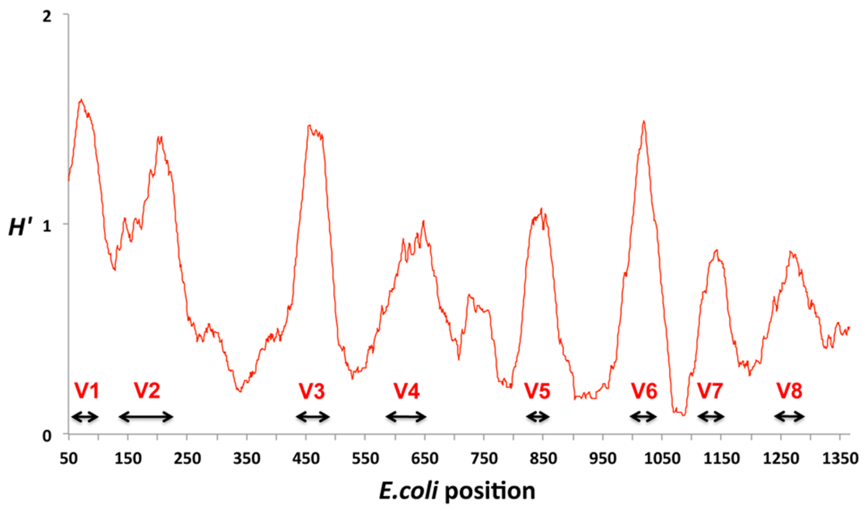

Figure 1: Across the 16S gene, some regions are "variable" (regions "V1" through "V9") relative to the surrounding "conserved" regions. The x-axis shows position in the 16S gene, measured by nucleotides as they appear in the *E. coli* genome. The y-axis (H') is "entropy", a measure of the variability in the nucleotides at that position in the gene. Vasileiadis *et al.* (doi:10.1371/journal.pone.0042671)

2.3 Getting DNA from a sample

After a sample is taken, the cells in the sample are lysed, typically using some combination of chemical membrane-dissolving and physical membrane-busting. The DNA in the sample is then *extracted*, meaning that all the protein, lipids, and other stuff in the sample is thrown away. From this pile of DNA spaghetti, we aim to collect information about the bacteria that were in the original sample.

[2]It's not unusual for bacteria to have multiple copies of the 16S gene, and those copies might not be identical to one another. Some people are concerned by the effect this could have on interpretations of 16S data (e.g., Kembel *et al.* [doi:10.1371/journal.pcbi.1002743] and Case *et al.* [doi:10.1128/AEM.01177-06]).

2.4 Amplifying the gene

In the amplicon sequencing approach, polymerase chain reaction (PCR) is used to amplify a section of the 16S gene. The size of the sequenced section is limited by the length of reads produced by high-throughput sequencing. The sections of the 16S gene that are amplified are named according to what variable regions of the gene are covered.

Of the nine variable regions, named V1 through V9, the region most commonly used in the microbiome literature is V4. However, different regions can provide different taxonomically resolution for different parts of the microbial tree of life. V4 is popular because it provides good taxonomic resolution for gut microbiota. By contrast, the an amplicon covering the first two regions (V1-V2) is more common in studies of the skin microbiome because those regions provide better taxonomic resolution for the microbes commonly found on the skin.

PCR reactions on these regions have primers that match the constant regions around the targeted variable regions. Papers should always say which primers they used, and they usually also mention the amplified region. The primers have names like 8F (i.e., a forward primer starting at nucleotide 8 in the gene) and 1492R (i.e., a reverse primer starting at nucleotide 1492).

2.5 The amplified DNA is not exactly like the original DNA

The process of extracting DNA from bacterial cells and then amplifying a 16S region introduces certain biases into the resulting sequence data. These effects mean that observed differences in bacterial community composition *between* samples collected in a single study are usually more reliable than apparent differences in composition *within* sample. In other words, 16S data are better able to support a statement like "species X is more common in patient cohort A than in to cohort B" rather than a statement like "species X is more abundant than species Y".

These biases also means that large effects, like variations over orders of magnitude, are to be trusted far more than smaller changes.

2.5.1 Extraction bias

Different cells respond differently to different extraction protocols. Using different extraction protocols on the same sample can produce markedly different results.[3] My takeaway is that, if you're comparing two different data sets, it's important to know if they used the same extraction methodology, since differences in the 16S data could be due to differences in the microbes or due to differences in the methods used to extract the DNA.

[3]E.g., Salter *et al.* (doi:10.1186/s12915-014-0087-z), Walker *et al.* (doi:10.1371/journal.pone.0088982), Rochelle *et al.* (doi:10.1016/0378-1097(92)90188-T), etc. But compare Rubin *et al.* (doi:10.1002/mbo3.216).

2.5.2 PCR bias

Although we say the PCR primers bind a "conserved" region, there is still variation in those regions. Thus, some bacteria in the sample will have different nucleotides at the primer binding site, meaning that the PCR primers will bind with different affinities to the DNA of different bacteria. This effect decreases the number of reads from bacteria whose constant regions don't match the primer.[4]

"PCR bias" encompasses other things beyond primer site binding bias. It's known that PCR has different efficiencies for different types of sequences, meaning that some 16S variable regions will amplify better than others. Also, statistical fluctuations can occur, especially in low-diversity samples. This means that a sequence that, by chance, gets lots of amplification in early PCR cycles could dominate the sample in late PCR cycles.

In general, PCR bias is more pronounced when the density of bacteria in the original sample is low, such that the PCR needs to be run for more cycles. For example, samples of human stool, in which the density of bacteria is enormous, are experimentally less finicky than skin swabs, which typically have very lower bacterial concentrations.

2.5.3 Chimeras

PCR also creates weird artifacts called *chimeras*. When using PCR to amplify two DNA sequences a and b, you'll get many copies of a, many copies of b, and some sequences that have an a head and b tail (or vice versa). If that chimeric a-b sequence looks like a real bacterial sequence, it can confuse downstream analyses.

2.5.4 Lab-, study-, and batch-specific effects

There are also biases that arise from *any* DNA-based experiment, like the biases that result from the method of collection or storage. There are many studies exploring how storage at different temperatures, storage for different lengths of time, different storage buffers, and so forth affect the measured bacterial community compositions. Regardless of what method of collection and storage is used, using the same methodology for every sample in a study is a crucial tool for reducing biases in the data.

2.6 Multiplexing

Next-generation sequencing became more helpful to microbial ecology when *sample multiplexing* (or "barcoding") was worked out in the early 2000s.[5] Before multiplexing, every sample had to be run on its own sequencing lane. This was expensive and bioinformatically annoying, since, especially in those early days

[4]It may be that there are a lot of interesting bugs whose 16S sequences are so divergent that they don't match the typical primers (cf. Brown *et al.*; doi:10.1038/nature14486).

[5]Cf., e.g., Binladen *et al.* (doi:10.1371/journal.pone.0000197).

of sequencing, it was often hard to distinguish a bad lane from a very unusual sample.

Multiplexing, by contrast, adds a *barcode* (or “tag”) to the 16S amplicon. Each barcode corresponds to a sample, and all amplicons in that sample get that barcode. It’s now common to multiplex 96 (or 384) samples and sequence them all in one lane.

Aside from making the sequencing 96-fold cheaper, multiplexing means that it’s easier to include some controls in each lane. Negative controls usually just vehicle with no DNA as a way to check for contamination from reagents or poor sample preparation. Positive controls typically take the form of mock communities of known composition, which can be used to check that the sequencing was not “weird”. If you have a lot of samples from the same project and you need to run them in more than one lane, you can use the positive controls as an internal check that sequencing proceeded similarly across lanes.

2.7 Sequencing

A little more work has to be done before putting the sample in the sequencer. These steps will depend on the sequencing platform. Here I’ll talk about Illumina because it’s popular[6] and I have experience with it. If you’re using a different sequencing platform, like Nanopore, then you’ll need to learn about the quirks of that platform elsewhere.

Samples to be sequenced on an Illumina machine need to have Illumina-specific *adapters* added in a third PCR (one for amplification, one to add the barcodes, and one to add the adapters). These adapters allow the DNA amplicons to bind the flowcell, where they are sequenced.

It is sometimes also desirable to have a *diversity region* added between the adapter and the 16S primer. The Illumina sequencers expect to see a diversity of nucleotides at every read position. In amplicon sequencing, almost all the reads are the same through the primer region, which can cause difficulties for the sequencer.

All of these pieces —the 16S region you’re interested in, forward and reverse primers, barcodes, diversity region, and Illumina adapters— are all made into a single *PCR construct*, which is a single piece of DNA. The sequencer reads the nucleotides in the construct and uses its knowledge about the arrangement of the construct to infer which nucleotides are the region of interest and which are the barcode.

[6] It wasn’t that long ago that 454 (or “Roche”) sequencing led the next-generation field. Plenty of papers used “pyrosequencing” (the technical word for 454’s sequencing methodology) as a synonym for “next-generation sequencing”.

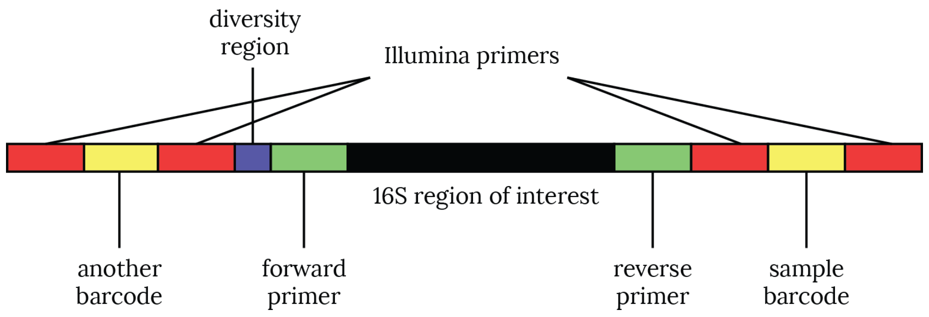

Figure 2: An example PCR construct.

3 16S data processing

As the microbiome field has matured, there are ever-better tools and pipelines for processing 16S data. These resources can save you time and can enhance reproducibility, but they are no substitute for a deep understanding of the underlying processes. This chapter steps through the files and algorithms used in the more mundane steps of 16S processing.

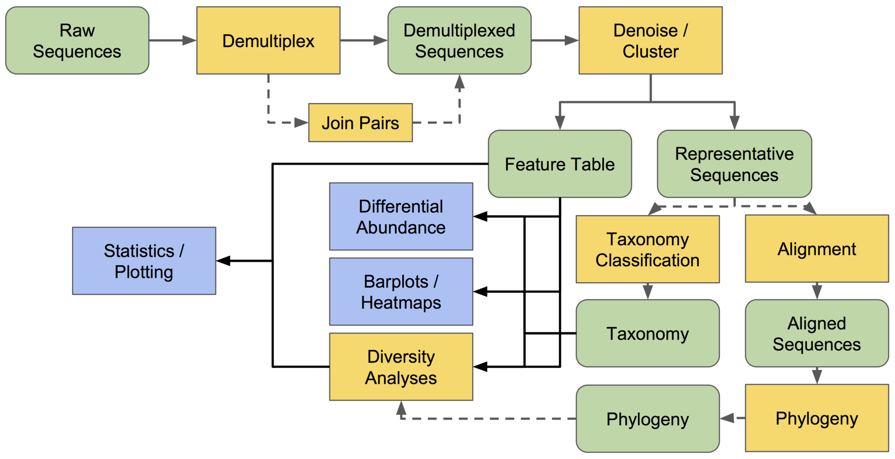

Figure 3: Overview of the 16S data processing and analysis process. Note the nonlinearities in the process; there may be more than one way to proceed through data processing and analysis. Reproduced from QIIME 2 documentation.

3.1 Phase I: Preprocessing

Before talking about how to preprocess the data, it's important to have a handle on what exactly we're talking about when we say "the data".

The two important file formats for 16S data are *fastq* and *fasta*.[7] Fastq is an Illumina-specific raw data format,[8] while fasta is the industry-standard way to display processed sequence data.

3.1.1 Fastq format

Fastq files usually have the extension `.fastq` or `.fq`. They are made up of *entries* that each correspond to a single read. Every entry is four lines; a well-formed fastq file has a number of lines that is a multiple of four. The lines in the entry are:

1. The *header* line
2. The *sequence* line
3. The *plus* line
4. The *quality* line

The header line must begin with the character `@`. The content of the rest of the line depends on the Illumina software version, but in general it gives information about the read: the name of the instrument it was sequenced on, the flowcell lane, the position of the read on the cell, and whether it is a forward or reverse read. In some version of Illumina, the barcode read is in the read's header.

Like the name suggests, the sequence line is a series of letters encoding the sequencing data, one character per nucleotide. Note that the letters might not all be `ACGT`. Other letters indicate that the nucleotide might be any of a group of bases. For example, `R` means purine (`A` or `G`). `N` means "no idea, any base possible".[9]

The third line must begin with the plus-character `+`. The rest of the line after the plus can be anything; it is often left blank.[10]

[7] Technically, these formats should be written FASTQ and FASTA, but I find this cumbersome, so I write them as if they were not acronyms. Fastq is pronounced "fast-Q". Fasta is supposed to be pronounced as "fast-A", but I hear "fast-uh" just as often.

[8] In what follows, I'll talk about "raw data", by which I mean data that you would get from the sequencing center. For Illumina data at least, there is actually a more raw kind of data that comes right out of the sequencing machine that gets processed right away. The software that processes that very raw data changes slightly across different version, so be prepared for slight variations in the format of your "raw" data.

[9] These other options are the IUPAC nucleotide abbreviations. There is a one-letter code for every possible combination of the four nucleotides.

[10] The original fastq specification (doi:10.1093/nar/gkp1137) allowed the sequence and quality information to run over multiple lines like in the fasta format. This led to a lot of confusion, since `+` and `@` appear in some quality encodings. It's now recommended to *not* spread the sequence and quality information over multiple lines. Thus, the plus line is there for backward compatibility.

The quality line gives information about the quality of the base calls shown in the sequence line. Each character gives information about the quality of one base call. Confusingly, the encoding has changed in overlapping and sometimes non-redundant ways.[11] In the newest Illumina format, the encoding goes, from low quality to high quality:

```
!#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
```

The letter `I`, the highest-quality mark, means that there is a $10^{-4.0}$ probability that this base is wrong (i.e., a 99.99% chance that it is correct). `H`, the second-highest quality, means that there is a $10^{-3.9}$ chance; `G` means $10^{-3.8}$, and so on.[12] The exclamation point is special: it means a quality of zero, i.e., that the sequencer has no idea what that base is.[13]

Here's an example fastq entry. The sequence and quality lines are too long to fit on the page, so I cut out some letters in the middle and put those dots instead:

```
@MISEQ578:1:1101:15129:1752#CCGACA/1
TATGGTGCCAGCCGCCGCGGTA...GCGAAGGCGGCTCACTGGCTCGATACTGACGCTGAG
+
>1>11B11B11>A1AA0A00E/...FF@@<@FF@@@@FFFFFFEFF@;FE@FF/9-AAB##
```

Some things to note about this read:

- The first line gives information about the machine used (MiSeq) and where the read was on the flow cell. `CCGACA` was the barcode read, and the final `/1` means it was a forward read.
- The plus line has been left blank.
- The quality of the read's base calls decreases from `>`, encoding quality 29 (i.e., a $10^{-2.9} = 0.1\%$ probability of error), all the way down to `#`, or quality 2 (i.e., a $10^{-0.2} = 63\%$ probability of error).

3.1.2 Fasta format

Fasta files usually have the extension `.fasta`, `.fa`, or `.fna`.[14] They are made up of *entries* that each correspond to a single sequence. Each entry consists of a

[11] Technically, these different encodings are named by their ASCII "offsets". ASCII is a system that associates individual characters with integers. The system I show here has an ASCII offset of 33: the character `I` has an ASCII value of 73, and the offset means you subtract the offset 33 from the ASCII value 73 to get the quality score 40. The other common offset is 64: in that case, the character `I` encodes quality 9. Illumina used a 64-offset for a while, but newer machines use a 33-offset.

[12] This conversion between quality Q (the integer) and the probability p of incorrect calling is the "Phred" or "Sanger" system: $Q = -10 \log_{10} p$. There is at least one other way of converting between Q and p, the "Solexa" system.

[13] Even more confusingly, in some versions of Illumina, the very low ASCII scores were used to mean special things. In the older, 64-offset Illumina encoding, quality 3 (character `C`) was the lowest possible, and the quality 2 character (character `B`) was instead used to show that the Illumina software had done its own internal quality trimming and had decided that, starting with the first `B`, that the rest of the sequence was "bad".

[14] The last `a` in `fasta` stands for "all", meaning nucleotide or amino acid or whatever. The `n` in `fna` stands for "nucleotide".

header line and one or more *sequence* lines. The *header* line must start with the greater-than symbol `>`. This signals the start of a new entry. The rest of the header line is an identifier, analogous to the content of the header line in a fastq.

In a fastq file, each linebreak was meaningful. In a fasta file, only the linebreak at the end of the header line is meaningful. Linebreaks in the following sequence lines are all ignored. The sequence for this entry just keeps on going until you hit another `>`. The early recommendation was to wrap all lines at 80 characters to make them easy to read on old-school terminals. Many people and software tools conform to this recommendation; others make each entry just two lines, one header and one sequence.

Fasta files provide strictly less information than fastq files, so normally you'll move to a fasta format after quality filtering, when you've decided you aren't going to use the quality data again.

Here's an example fasta entry in the traditional format:

```
>sequence1
TACGGAGGATCCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGAGCGTAGGCGGGTTGT
TAAGTCAGTTGTGAAAGTTTGCGGCTCAACCGTAAAATTGCAGTTGATACTGGCATCCTT
GAGTACAGTAGAGGTAGGCGGAATTCGTGGTGTAGCGGTGAAATGCTTAGATATCACGAA
GAACTCCGATTGCGAAGGCAGCCTGCTGGACTGTAACTGACGCTGATGCTCGAAAGTGTG
```

3.1.3 Finding your raw data and metadata

Before trying to process your dataset, be sure you have the appropriate raw data and metadata. In all cases, this means you'll need at least one set of reads. For Illumina, this means a *fastq* file of forward sequences might have a name like:

```
130423Alm_D13-1939_R1_sequence.fastq
```

This filename has information about the date of sequencing (2013-04-23), the group that requested the sequencing (Alm Lab), and some specifics about the sequencing run. The `_R1_` indicates that these were forward reads.

If you did paired-end sequencing, you will also need the *reverse reads*. This is a fastq file with the same filename as the forward reads except with `_R2_` in it. Every entry in the reverse reads should match an entry in the forward reads.

3.1.4 Demultiplexing

As mentioned earlier, multiple samples are *multiplexed* so that they can be sequenced in one sequencing lane. Now that sequencing is complete, the samples must be *demultiplexed*, which means that the reads from the one sequencing lane are divided up into their corresponding samples. It is now standard practice for delivered to you to have already been demultiplexed. Rather than all the forward reads from a sequencing run being in one big file, they are split, with one file per sample. Reverse reads are similarly split so that each sample should have two associated read files, one forward and one reverse.

If you data are not demultiplexed (i.e., you only have one big file of forward reads), then you will also need the *barcode* or *index reads*. Depending on your Illumina version, this information might be in different places. In some datasets, it's in the file with the forward reads. For example, in the fastq entry above, the header line contains the barcode read `CCGACA`. In other datasets, you might find the index reads in a file with a name that has `_R3_`, `_I_`, or `_I1_` in it. You will also need a *barcode map* that links each barcode with each sample.

In theory, demultiplexing sounds simple: you look in the raw sequence for the barcode, look up what sample that barcode corresponds to using the barcode map, and then trim off the barcode. In practice, there are two questions that need to be answered:

- *Which of the known barcodes is the best match for this barcode read?* That's a pretty straightforward answer. But what if there's a tie? In fact, barcodes are chosen with an error-correcting code[15] so that a tie implies that you have at least two errors in the read, which is unlikely.
- *Is the match with the known barcode good enough?* A common approach is, given a barcode read, to compare that read with all the known barcodes (i.e., the barcodes you're looking for). If the known barcode that matches best has more than one mismatch with the barcode read, call that read "bad" and discard it.

3.1.5 Removing primer sequences

During PCR amplification, it is the primer that is amplified, not the DNA that the primer was bound to. This means that, if there was a mismatch between the primer and the DNA of interest, you won't see the actual DNA; you only ever see the primer sequence. Thus, keeping the primer sequences in your reads makes it appear in downstream analyses as if the primer sequence was the actual sequence present in the sampled DNA. The common practice is to cut off the primers, and nowadays primers may have already been removed (or "trimmed") before they are handed to you, the researcher.

If the primers haven't already been removed, you'll need to remove them yourself. In an ideal world, this is straightforward: you find the piece of your read that matches the primer, and you pop it off. In practice, there are two important considerations:

- *Where do you look for the primer?* Does the primer start at the very first nucleotide of the read, or a little further in? You can put a lot of flexibility in this step without a lot of negative effects, but it's good to know what's going on in your data.
- *What does "match" mean?* How many mismatched nucleotides do you allow between the read and your primer sequence before you consider the read "bad"? For example, you might discard the reads that have more

[15]E.g., Hamady *et al.* (doi:10.1038/nmeth.1184)

than one mismatch in the primer sequence.

There are also some less intellectually interesting "gotcha's" that I suffered multiple times as a graduate student:

- I didn't know the primer sequences, or the primer sequences I was told were not the actual ones present in the data.
- I thought I had the sequences of the primers but I actually had the reverse complements of those primers.
- I thought (or had been told) that the primers had been pre-removed when they were actually still present.

3.1.6 Summary of expected files

All told, you will likely need:

- Demultiplexed forward reads, with one file per sample
- Demultiplexed reverse reads, with one file per sample

However, depending on how the data were delivered to you, you might also need:

- Non-demultiplexed forward reads (i.e., one big file of forward reads)
- Non-demultiplexed reverse reads
- Barcode reads
- Barcode map
- Primer sequences

If you download a dataset or get it from a collaborator, it's important to understand which of these preprocessing steps have already been performed.

3.2 Phase II: Cleaning

These steps take the raw data and turn it into biologically relevant stuff. There is some freedom about the order in which they can be done. I separate these steps out from preprocessing because the choices you make here can more substantially affect your data.

3.2.1 Quality filtering

Sequences tend to vary in overall quality —some good, some bad— and the number of bases they have that are good. Inherent in the Illumina technology is a trend for sequences to decrease in quality as you move along the read.

The sequencer will give you a sort of quality report about your sequences' average quality. It will give you a sense of whether your sequencing run as a whole was good, and it will give you a sense of whether you got the sort of good-quality length you were hoping for. The big quality report gives you a sense of whether you should do the whole sequencing run over again.

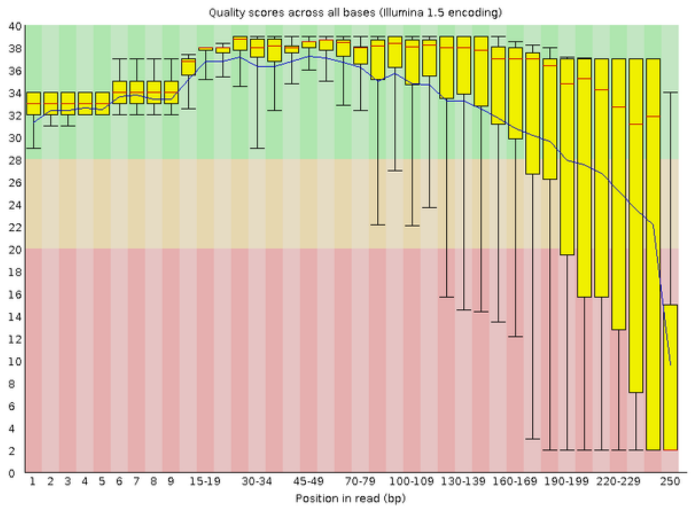

Figure 4: A read quality report delivered by the Illumina software.

Even in a good sequencing run there are bad sequences that should be filtered out. There are two common ways to quality filter:

- *Quality trimming* means truncating a read at some nucleotide after which the read is "bad".[16] A common approach is to truncate everything after the first nucleotide whose quality is below some threshold.
- *Global quality filtering* means discarding an entire read if the average quality of the read is too low. Maybe no individual nucleotide falls below your trim threshold, but the general poor quality of the read means that you'd rather not include it in analysis. This criterion is expressed equivalently as "average quality" or "expected number of errors".[17]

3.2.2 Merging

When doing paired-end sequencing, it's desirable for the two reads in the pair to overlap in the middle. In that case you can *merge* (or "overlap", "assemble", or "stitch") the two reads into a single full-length read whose quality in the middle positions is hopefully greater than the quality of either of the two reads that produced it.

Merging requires answering a few, fairly complex questions:

[16]Confusingly, "trimming" also refers to a different process when, if you're doing unpaired amplicon sequencing, you pick a length, discard all reads shorter than that, and truncate all the longer sequences at that length. It is essential to do this when using certain *de novo* OTU calling methods, and it's probably beneficial to do with reference-based OTU calling and or taxonomy assignment methods.

[17]Edgar & Flyvbjerg (doi:10.1093/bioinformatics/btv401)

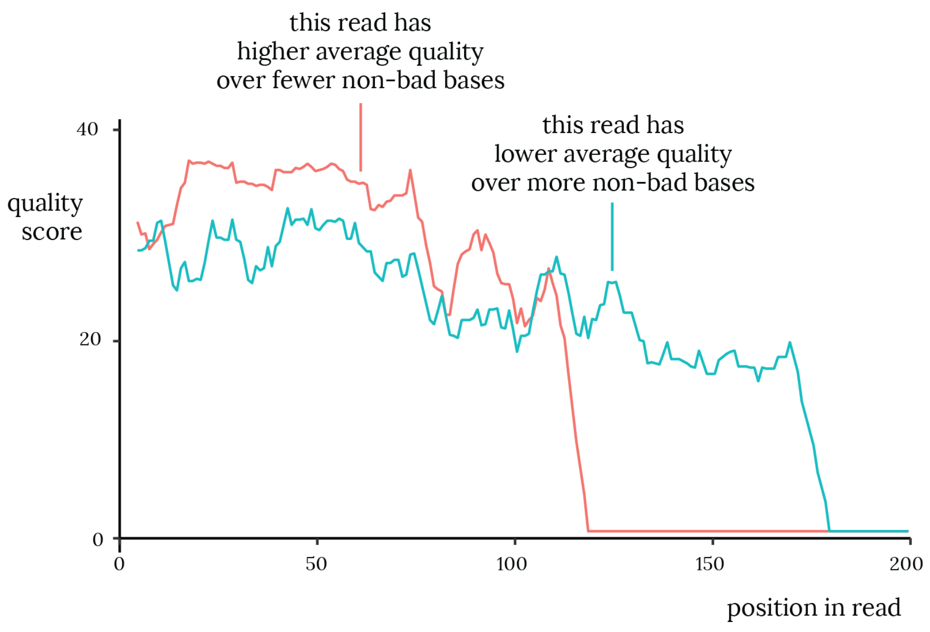

Figure 5: An example of two reads differ in overall quality and number of "non-bad" reads.

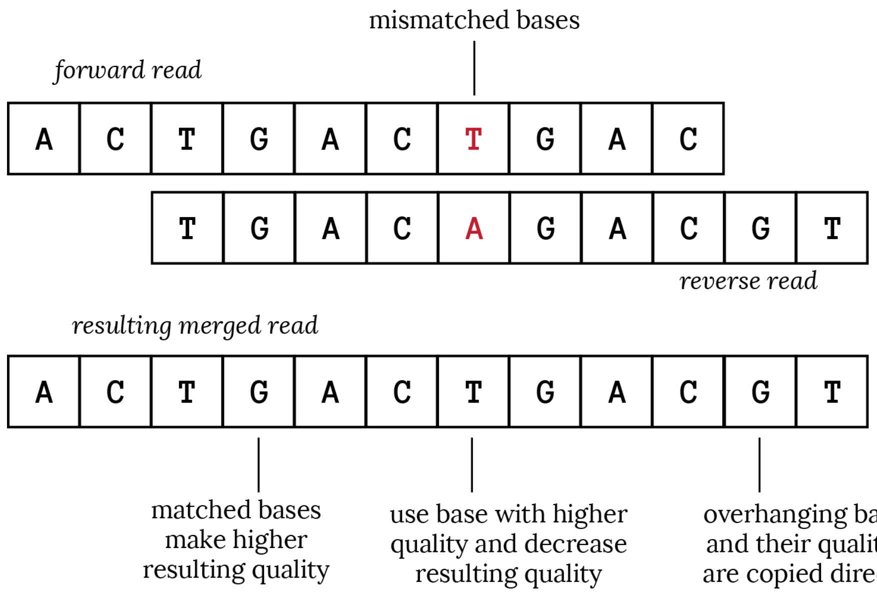
mismatched bases
forward read
A C T G A C T G A C
T G A C A G A C G T
reverse read
resulting merged read
A C T G A C T G A C G T
matched bases make higher resulting quality
use base with higher quality and decrease resulting quality
overhanging bases and their qualities are copied directly

Figure 6: Merging aligns reads, makes a new sequence, and computes new quality scores. Adapted from the `usearch` manual (drive5.com/usearch/manual/merge_pair.html).

- *What is the best position for merging?* If you were sure all your amplicons are exactly the same size, then this is trivial: just overlap them at the right length. However, even in amplicon sequencing, there are insertions and deletions in the 16S variable regions, so we can't be sure that all merged reads will be the exactly the same length.
- *Is the best position good enough?* If you have two reads that don't overlap at all, should you even include it in the downstream analysis?[18] How good is good enough?
- *What are the quality scores of the merged nucleotides?* This requires some Bayesian statistics.[19]

3.3 Phase III: Denoising (and chimera removal)

Denoising is the process of accounting for errors inherent in sequencing technology, especially the Illumina platform. In short, denoising "corrects" sequencing error, decreasing the diversity of sequences in the data that are due to technological error rather than to true biological diversity. Two important implementations of denoising are DADA2 and Deblur.[20]

The work of denoising was previously done as a part of operational taxonomic unit (OTU) calling. Denoising and OTU calling are sufficiently conceptually complex and historically intertwined that I will discuss them separately in the next chapter.

This step typically also includes *dereplication*. Because there are fewer unique *sequences* (strings of `ACGT`) than there are *reads*, it makes more sense to keep a list of unique sequences and a table of how many times each sequence appears in each sample.

3.3.1 Chimera removal (or "slaying")

As mentioned early on, PCR can produce chimeric sequences.[21] Depending on your choices about OTU calling, you may want to remove chimeras after dereplicating. Chimera removal checks to see which of your dereplicated sequences can be made by joining the first part of one sequence (the "head") with the last part of another sequence (the "tail").

Chimera removal methods come in two main flavors: *reference-based* and *de novo*. In reference-based methods, you look for the head and tail sequences in some database. Popular databses include Greengenes, SILVA, the Broad

[18]If you had two paired-end reads that didn't overlap but you were somehow sure of the final amplicon size, then you could insert a bunch of `N`'s in between.

[19]Rodrigue *et al.* (doi:10.1371/journal.pone.0011840); Edgar & Flyvbjerg (doi:10.1093/bioinformatics/btv401)

[20]Callahan *et al.* (doi:10.1038/nmeth.3869); Amir *et al.* (doi:10.1128/mSystems.00191-16)

[21]The Chimera was a monster in Greek mythology. It had the head of a lion and the tail of a snake. It was slain by the hero Bellerophon.

Institute's ChimeraSlayer (or "Gold") database, and the Ribosomal Database Project (RDP) database.

In *de novo* methods, you ask which of your sequences have could be generated by combining other (typically more abundant) sequences from that same dataset. In the past, this was a computationally-expensive undertaking, but there are ever-improving methods, notably, the UPARSE[22] algorithm.

3.4 Phase IV: OTU calling

As mentioned above, calling (or "picking") operational taxonomic units (OTUs) is a conceptually and historically complex topic, so I will treat it in a separate chapter. In short, OTU calling assigns every dereplicated sequence to a group, or OTU. Just as dereplication produces a list of unique sequences and a table indicating how many times each sequence appears in each sample, OTU calling produces a list of OTUs' *representative sequences* and an *OTU table* that shows the number of times each OTU appeared in each sample.

3.4.1 Phase V: Analysis

The part where you actually use your data! Analysis is outside the scope of this work.

4 Denoising and OTU calling

As mentioned in the last chapter, denoising and OTU calling are complex and intertwined topics, so I discuss them separately here.

4.1 An abridged history of the OTU

In the 1980s, Carl Woese showed that the 16S gene could be used as a molecular clock. Using 16S data, he re-drew the tree of life, breaking up the older Monera into Bacteria and Archaea.[23] The 16S gene was therefore a promising practical candidate for distinguishing bacterial species.[24]

The species concept is easy to define for sexual macroorganisms: two living things of opposite sex are in the same species if they can produce fertile offspring together. Bacteria don't have sex, but they do perform homologous recombination. Homologous recombination requires some sequence similarity, so it came

[22]Edgar (doi:10.1038/nmeth.2604)

[23]Woese, Kandler, Wheelis (doi:10.1073/pnas.87.12.4576)

[24]In 1991, PCR-amplified portions of the 16S gene were used to identify known species (Weisburg *et al.* [doi:10.1128/jb.173.2.697-703.1991]). The paper has a prescient final sentence: "While [PCR] should not be a routine substitute for growing bacteria, picking individual colonies, and confirming their phenotypic and biochemical identities, it will enable experiments to be performed that were not previously possible."

about that a common definition of a bacterial species was all those strains whose isolated DNA was 70% DNA-DNA-hybridization similar.[25]

In the 1990s, people sequenced the 16S genes of the strains that had been grouped into species by the hybridization assay. It emerged as a rule of thumb that two bacteria were the same species if their 16S genes had 97% nucleotide identity. Because of this history, a lot of discussion around OTUs involves finding 97% clusters, and "OTU" was often used as a shorthand for "97% clusters".

Personally, I think the word "operational" in OTU should mean an OTU is whatever group of sequencing reads you choose to be the unit of analysis in your downstream work. This, however, is not the standard nomenclature. I'll also note that there are various other terms like "phylotype" and "oligotype"[26] used for groupings related to but conceptually distinct from "OTUs".

4.2 Denoising

Historically, people called OTUs for a few reasons. One very practical reason was data reduction. Dereplication can give you hundreds of thousands of unique sequences, and doing analysis with hundreds of thousands of unique units was computationally infeasible in the early days of 16S sequencing.

A second important and related reason was *denoising*. Imperfect technology means that one can never really be sure whether two reads have different sequences because of technological error or because those two reads arose from biologically different sequences. One way to denoise data was to say that we couldn't really trust our equipment to be able to distinguish sequences that were less than about 97% similar anyway, so we might as well lump those sequences into one unit for analysis.

Today, we use purpose-built, highly sophisticated denoisers. To give a sense of how denoisers work, consider this hypothetical example. Say you are sequencing an amplicon that has only 10 nucleotides, and the sequencer has an error rate of 1%, meaning that each base pair has an independent, 1% chance of being misreported. Some straightforward math shows that there is a 90% chance that all 10 nucleotides will be correctly reported, a 9.1% chance that 1 nucleotide will be incorrect, and a 0.9% chance that 2 or more nucleotides will be incorrect.

In this example, if there was only one true sequence in the biological sample, then we would expect 90% of reads to be that true sequence, while the remainder would be split among the various erroroneous sequences. For example, of the 9.1% of reads with one error, one in ten (i.e., 0.91% of the total reads) would have that error in the first nucleotide, and one-third of those reads (i.e., 0.3% of the total reads) would have each of the 3 incorrect base pairs in it.

[25]Konstantinidis, Ramette, Tiedje (doi:10.1098/rstb.2006.1920). To this day there is still a large debate about the microbial species "concept".

[26]Eren *et al.* (doi:10.1111/2041-210X.12114)

Table 1: Hypothetical example in which there is a single 10 nucleotide sequence present in the sample DNA, a 1% error rate, and an even distribution among errors.

Percent of reads	Read sequence
90%	GACAGGTACA
0.3%	**A**ACAGGTACA
0.3%	**C**ACAGGTACA
0.3%	**T**ACAGGTACA
0.3%	G**C**CAGGTACA
0.3%	G**G**CAGGTACA
0.3%	G**T**CAGGTACA
...	...

On the other hand, if one of these sequences that could be explained as error was more frequent than was expected given the statistical model for errors, then we would conclude it is also a real, biological sequence. For example, if the first erroneous sequence (i.e., AACAGGTACA) appeared as often as the original sequence (GACAGGTACA), then we would expect that they are both meaningful. However, our ability to distinguish real biological variations from technological error is limited by the abundance of these potentially erroneous sequences and the sequencer error rate: if the error rate is high or the true variant sequence is rare, we will not be able to determine that it is not simply an erroneous read.

Denoisers infer which observed read sequences are likely erroneous deviations from other, more abundant, "true" sequences. In this way, denoisers can distinguish between similar sequences that are likely to be truly biologically distinct, in ways that 97% OTU calling cannot.

The output of denoising algorithms are called *amplicon sequence variants* (ASVs), which are estimates of the true sequences that were presented in the original DNA. The potentially substantial reduction in numbers of unique sequences that comes from denoising, combined with improved computing power, means that, in many cases, calling OTUs is no longer a practical necessity. However, because the OTU concept dominated 16S data processing for so long, it is still common to say that an analysis uses "100% identity OTUs", which simply means that each sequence or ASV was treated as its own OTU. In other words, "100% OTUs" means that OTUs weren't called at all!

4.3 Philosophical reasons for OTU calling

Although it is no longer stricly necessary to call OTUs, there are still philosophical[27] and analytical for doing so, depending on your study's purpose. If you

[27] Jax (doi:10.1086/506237) reviews the different ways ecological units are viewed from ontological and functional perspectives.

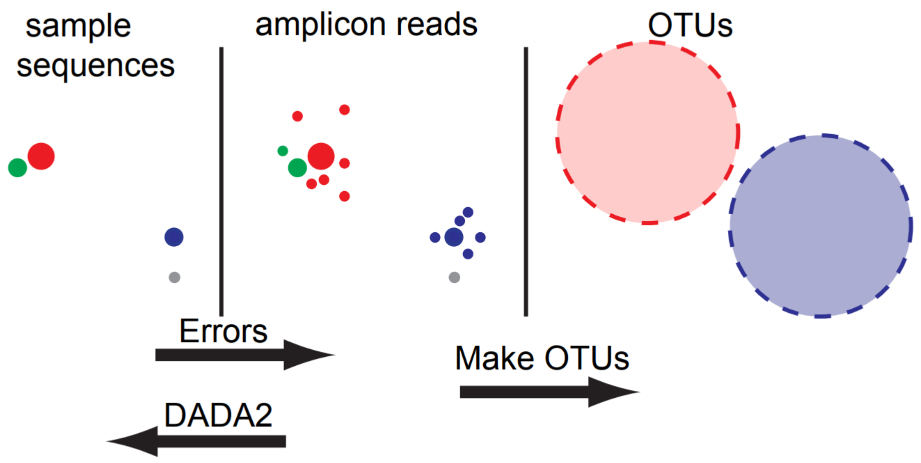

Figure 7: Denoising algorithms like DADA2 correct the original sequencing data using statistical error models. OTU calling, by contrast, simply groups similar sequences. In this illustration, a denoiser can eliminate erroneous variants (the small dots) while also distinguishing two similar but biologically distinct sequences (red versus green). OTU calling removes the erroneous sequences at the cost of lumping together meaningfully different sequences (red with green, dark with versus light blue). Reproduced from Callahan *et al.* (doi:10.1038/nmeth.3869).

want to study bacterial species and are a firm believer in the idea that a 97% cluster is the best approximation of a species, then you'd want to organize your data into those approximate-species and go from there.

More generally, you'll want to organize your sequences into some operational unit (i.e., OTU) that works well with the kind of analysis you want to do. For example, to examine very broad broad changes in community composition, you might want to call OTUs that are your best approximations of phyla. If you're interested in what individual organisms are doing, you'll probably want to do very little (if any) grouping of sequences into OTUs, since the unique sequences are, in a sense, the best information you have about those organisms.

Clustering your sequences into a few, large OTUs can make it easier to think about your data, but beware: you want those clusters to be meaningful. You want them to be the called in the way that makes them most useful for answering the question you want to answer.

4.4 OTU-calling methods

There are multiple ways to group sequences into OTUs. Here I review the most important ones, in the order I thought was easiest to explain.

4.4.1 Amplicon sequence variants, or 100% identity OTUs

As discussed above, this approach, which was originally infeasible because of computational and technical reasons, is becoming increasing popular. Lower error rates from sequencers combined with sophisticated denoising algorithms means that distinct amplicon sequence variants (ASVs), even ones that differ from one another by only one nucleotide, may very well be biologically distinct.

4.4.2 *De novo* clustering

As the name suggests, *de novo* clustering means making your own OTUs from scratch. There are many approaches to *de novo* clustering methods, but they all follow the same basic principle: they try to identify a set of OTUs that are at some maximum dissimilarity relative to one another. (As you might guess, 97% OTUs are popular.) Every OTU will be assigned a *representative sequence*, which is either one of the OTU's member sequences or a composite sequence based on the members.

De novo clustering suffers from some insidious and very serious disadvantages. First, *de novo* methods are more computationally expensive than other methods. Second, it is becoming increasingly clear that many methods produce *de novo* OTUs that are not *stable*, meaning that small changes in the sequence data you feed into the algorithm can lead to large changes in the number of OTUs, the OTUs' representative sequences, and the assignment of reads to OTUs. Third, it is difficult to incorporate new data into a dataset that has been processed into *de novo* OTUs. It usually requires calling OTUs all over again. It's also difficult to compare *de novo* OTUs across datasets: you and I might have lots of the same sequences, but our *de novo* OTUs might differ.

The principle advantage of *de novo* clustering is that it won't throw out abundant sequences from your data.

4.4.3 Reference-based methods

In reference-based OTU calling, the OTUs' representative sequences are specified ahead of time in a database like Greengenes. Each sequence in your query dataset is assigned to one of the sequences in the reference dataset based on their sequence similarity. Greengenes has been such a popular reference database that I often heard "OTU" used to mean "the 97% OTUs in Greengenes", although that usage is becoming less common as denoising becomes more popular.

The principle advantages of reference-based calling are:

- *Stability.* Similar inputs should produce similar outputs, since you're just comparing to a fixed reference.
- *Comparability.* If you and I called our OTUs using the same reference, it's easy for us to check if we have similar sequences in our datasets.
- *Computational cheapness.* Unlike *de novo* OTU calling, reference-based methods only need to hold one sequence in memory at a time, so they

require less RAM and are easy to parallelize.
- *Chimeras need not be slain.* If you're only keeping sequences that align to some database, which has hopefully been pre-screened for chimeras, then you don't need to worry about them yourself.

The major weakness of reference-based methods is insidious: if a sequence in your query dataset doesn't match a sequence in the database, what do you do? Frighteningly, many methods just throw it out without telling you. If you work in the human gut microbiome, this might not bother you, since the gut is the best-studied ecosystem, and gut bacteria are well represented in databases like Greengenes. However, if you work in environmental microbiology or even in mice, many of the sequences in your studied bacterial community might not be sufficiently similar to a sequence in Greengenes to be assigned to a reference OTU.

Reference-based methods also suffer a converse problem: what if your sequence is an equally good match to more than one database entry? This can happen in amplicon sequencing: the Greengenes OTUs are the entire 16S gene (about 1400 bases), but you only have a little chunk of it (say, 250 bases). The Greengenes OTUs are, say, 97% similar (i.e., 3% dissimilar) across the *entire gene*, but they might be identical over the stretch that aligns to your little chunk.

4.4.4 Open-reference calling

The process I described above —just throwing out non-matching sequences— is called *closed-reference* calling. If you're interested in those non-matching sequences, you could gather them up and group then into *de novo* clusters, then combine your reference-based OTUs with your *de novo* OTUs. This mix of reference-based and *de novo* calling is named *open-reference* calling.

4.4.5 Taxonomy-based assignment

ASVs, 100% identity OTUs, and *de novo* OTUs can be hard to make sense of, since they are essentially one or more strings of `ACGT`. Reference OTUs also tend to have unilluminating names. For example, the Greengenes OTUs are labeled with numbers. To help interpret the raw sequences, it's very common to do *taxonomy* (or "lineage") *assignment*, which means determining a sequences domain, phylum, class, order, family, genus, and species.[28]

Taxonomy assignment requires comparing query sequences with a database of sequences that already have taxonomies associated with them. For example, the Greengenes database has taxonomies associated with all its OTUs.[29] Thus,

[28]16S rarely has sufficient resolution to identify species, and it often lacks information to determine placement at even higher levels. It is also worth noting that there are intermediate ranks (like subclass).

[29]The last update of the Greengenes database was in 2013. Bioinformatic methods have evolved since then, and the quality of the taxonomy assignments in the database is open to question. Cf., e.g., Yokono, Satoh, Tanaka (doi:10.1038/s41598-018-25090-8).

reference-based OTUs may automatically have taxonomies assigned to them.

Another popular method for taxonomy assignment is the Ribosomal Database Project's (RDP) naive Bayesian classifier.[30] Rather than comparing a sequence to existing OTUs, the RDP classifier breaks up the sequence into k-mers (all subsequences of the original sequence that have length k) and compares the k-mer content of that sequence to a database that links k-mer content to taxonomy. The practical advantage to this approach is that it gives *confidences* to each level of the taxonomic assignment. For example, a sequence might definitely be from some phylum (99%), but it might be difficult to specify its class (80%) and nearly impossible to identify its order (30%). In contrast, in the purely reference-based approach, the same sequence might happen to hit an OTU that is classified all the way down to the species, and you would mistakenly think that your sequence had a lot of taxonomic information in it.

Regardless of how taxonomy assignment is performed, it is important to remember that it is a distinct process from OTU calling. One can call OTUs without assigning taxonomies, or one can actually call OTUs by grouping sequences that are all assigned to the same taxonomy. Whether grouping sequences and doing analyses by taxonomic groupings constitutes re-calling OTUs is a question of semantics.

4.4.6 Distribution-based methods

The algorithms mentioned above mostly don't take notice of how those sequences are distributed among the samples. Preheim *et al.* (doi:10.1128/AEM.00342-13) showed that you get OTUs that better reflect the composition of a known, mock community if you take the sequence provenances into account. If an abundant sequence and a sequence-similar, rare sequence are distributed the same way across samples, the rare sequence is probably sequencing error and should be put in the same OTU with the abundant one. Conversely, if two very similar sequence are never found together, they probably represent ecologically-distinguishable bacteria, so they should be kept in separate OTUs. This approach is called *ecologically-based* (or "distribution-based") OTU calling.

4.5 How many OTUs?

A pet peeve of mine is when someone asks "how many OTUs" were in some sample. That number, on its own, means very little; it matters how the OTUs were called. Asking "how many OTUs" is like asking how many kinds of books I read last year. Do I answer "two", for fiction and nonfiction? Or do I put each book into its own category, because each one had its own author?

[30]Wang *et al.* (doi:10.1128/AEM.00062-07)

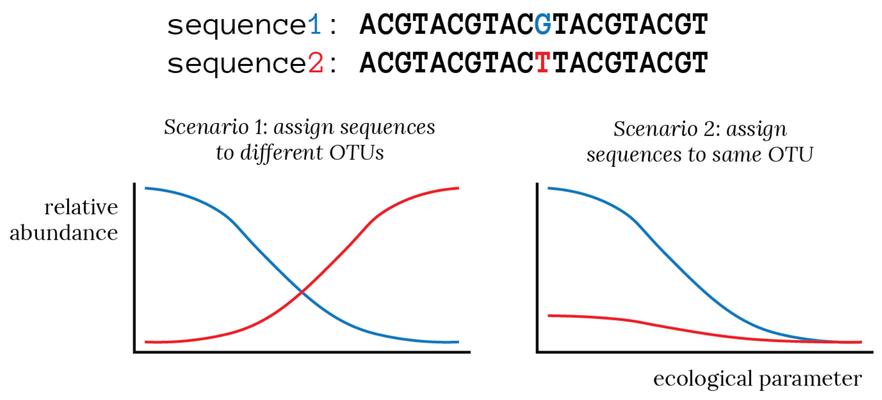

Figure 8: Distribution-based OTU calling separates similar sequences if they are distributed differently across samples. Adapted from Preheim *et al.*

5 Using QIIME 2

In previous versions of this primer, I encouraged readers to develop their own scripts for 16S data processing. I did this in part because the two, most user-friendly pipelines at the time, QIIME 1 and mothur, had characteristics that left me very unsatisfied. The field was also younger, and the methodology was more in flux, so it seemed wiser to get closer to the nuts and bolts of things. Now, however, I enthusiastically recommend using QIIME 2 for 16S data processing.[31]

QIIME 2 has extensive and ever-improving online documentation. Rather than repeat that material here, I will emphasize a few important points about working with QIIME 2.

5.1 QIIME 2 consists of plugins

The QIIME 2 documentation describes it as a "decentralized microbiome analysis package". It is a collection of individual tools, called *plugins*, with common interfaces. For example, there is a plugin called *quality-filter* that has a "method" *q-score* that filters 16S sequences in a fastq file based on quality scores. Another plugin *vsearch* has a method *cluster-features-de-novo* that does *de novo* OTU clustering. Every step in the 16S analysis process maps onto a method in one of the QIIME 2 plugins.

5.2 QIIME 2 packages data in artifacts

QIIME 2 methods do not run on human-readable data files like fastq's and fasta's. Instead, those human-readable files need to be *imported* into QIIME 2 *artifacts*. This importing process compresses the original data and also adds

[31]The two important changes for me were the plugin system and advances in virtualization software and virtual environment management.

some helpful metadata, called a *provenance*, which tracks the history of what methods have been run on a set of data.[32] If you want to directly interact with the data at any step in a QIIME 2 analysis, you will need to export it out of the artifact format.

5.2.1 QIIME 2 runs in compartmentalized computing environments

A challenge with a complex software project like QIIME 2 is that its software dependencies can conflict with other software dependencies. QIIME 2 avoids these problems using the modern solution of compartmentalized environments, either a virtual environment (via `conda` or a similar tool) or a virtual machine (such as Docker). As a user, this may introduce a greater learning curve to figure out how to initially install the software, but it will reduce downstream problems.

5.3 QIIME 2 still relies underlying algorithms and your parameter choices

The plugin concept means that QIIME 2's underlying algorithms are fairly transparent. This does not mean, however, that everything that nothing that QIIME 2 does will be surprising or, to put it bluntly, bad. QIIME 2 is only as good as its underlying algorithms and the appropriateness of the parameters you supply it.

For example, most pipeline software, QIIME 2 included, will allow many parameters to have default values. In many cases, these values will be appropriate for your data. In others, they may lead to very different results. The onus is on you as a researcher to understand the merit of the parameter values you selected, even if you selected the default ones.

As a second example, some of the methods in QIIME 2 rely on the USEARCH algorithm[33] to match query sequences to database sequences, such as when performing reference-based OTU calling. At the risk of making a mountain of a molehill, I dive a bit into the details of USEARCH to demonstrate the complexity that can underlie QIIME 2's pleasant user interface.

USEARCH is an algorithm used to match query sequences to database sequences. It is used, for example, to do reference-based OTU calling. One might expect that, given the same input sequence and the same database, the algorithm would also match that query sequence with the same database sequence. In fact, the picture is a little more complicated.

Critically, USEARCH is a *heuristic* algorithm. This means that it applies shortcuts to achieve faster speed:

1. Search for a match according to a decreasing expected sequence similarity.

[32]In the art world, *provenance* refers to the history of custody and ownership of a work of art. A reliable provenance is an important part of how you determine if a work of art is authentic.

[33]Edgar (doi:10.1093/bioinformatics/btq461)

2. Use heuristics to speed up the sequence alignments.
3. Apply stopping criteria. The default is to stop after a single hit that meets the accepting criteria.

The first shortcut means that your query sequence will be compared to database sequences according to some order; the shortcut rule means that the first of these database sequences that is a sufficiently good match to your query will be delivered as the result. For example, if you are assigning your OTUs by making a search to the 97% OTUs in Greengenes, then the first database sequence that is at least 97% similar to your query will be considered its parent OTU.

The third shortcut would be of no concern if the comparisons were performed in the order of decreasing sequence similarity, so that the first hit was always the best one. However, USEARCH needs to somehow guess, ahead of performing the actual alignment, which sequence in the database will be a good match to the query sequence.

USEARCH guesses the sequence similarity between two sequences using the U value (which is the U in USEARCH). U is the number of unique *words* shared by two sequences. You get these words by looking for length w (default is 8) subsequences starting at positions spaced μ apart. For example, if $\mu = 1$, then your words are all the w-mers in the sequence. If $\mu = w$, then your words are the first w nucleotides, the next w nucleotides, and so forth. Conveniently, sensible values for μ are inferred using tables of optimal choices derived from running USEARCH on databases of sequences using different values of the identity threshold.

This heuristic selection of database entries for comparison can lead to some quirky results, which I described in detail elsewhere.[34] In short, even reference-based OTU calling, which is expected to be stable and replicable, does not necessarily produce the results you might *a priori* think it should. Entire sequences and genera of bacteria can appear or disappear from your dataset depending on your choice of parameters for USEARCH.

The take-away is that a user-friendly interface cannot make the underlying algorithms user-friendly. The cutting edge is often sharp.

6 Parting wisdom

I hope this primer has given you exposure to fundamental concepts in 16S data processing. To give you a sense of where to go from here, I have some parting wisdom.

[34] Tsou *et al.* (doi:10.1080/19490976.2020.1747336)

6.1 Become more computationally proficient

QIIME 2 and other tools do provide some visualization and analysis capability, but if you want to deviate from the beaten path even a little bit, you will need to develop your own computation skills.

The first key skill to learn is the Unix command line and the basic commands like `cat`, `cd`, `cp`, `ls`, `mkdir`, `mv`, `rm`, `head`, `less`, `sed` (or `awk`), `wc`, `grep`, and `vi` (or `emacs`). These tools will help you when looking at raw data, using computational servers, or writing your own computer code.[35]

The second key skill is to learn a programming language with decent bioinformatics and statistics packages. Python and R are good choices. As a Masters student, after many years of fitfully using tools like Mathematica, I spent one week reading and doing all the exercises in Mark Lutz's *Learning Python*. It was one of the best investments I've ever made.

6.2 Be a data and algorithm detective

The best tool in your belt is a curious attitude. Be critical of your data at every step in the pipeline before you move onto the next step. Does it look the way you expect? How can you check? You may save yourself from substantial headache later on if you can catch bugs early in your analysis.

As mentioned earlier, processing 16S data requires many decisions, some small and seeming inconsequential, like error thresholds, and some large, like the choice of denoising algorithm of OTU picking method. Pipeline software like QIIME 2 can, through the setting of default values, create the illusion that some of these choices are not important, or even that you didn't need to make a choice. I only want to emphasize again that, in 16S data processing, the devil can be in the details.[36]

6.3 Get help

16S processing and analysis is a rapidly evolving and complex field. Even if you do develop strong computational skills, it is very wise to collaborate with experts.

Happy processing, and good luck!

[35]Maybe one day we'll have sexy drag-and-drop, hologram-style data processing for 16S, but for the foreseeable future it's going to look like the scene in *Jurassic Park* where Samuel L. Jackson is hunched over a computer muttering "Access main program... Access main security... Access main program grid..."

[36]Tsou *et al.* (doi:10.1080/19490976.2020.1747336)