12/22/14

Docherty Lab Illumina Mi-Seq Analysis Guide
Modified from MiSeq SOP: http://www.mothur.org/wiki/MiSeq_SOP

1. Unzip all the sequence files you want to analyze and put them into a folder on your desktop.
2. Put the mothur.exe executable file into this folder as well.
3. Copy and paste the silva.bacteria reference alignment (or greengenes alignment) into the folder
4. Create a file that contains the file names of all your sequences and call it stability.file.txt. The first column is the sample name, the second column is the .fastq file name of the forward read and the third column is the .fastq file name of the reverse read. See the example below:

F3D0 F3D0_S188_L001_R1_001.fastq F3D0_S188_L001_R2_001.fastq
F3D1 F3D1_S189_L001_R1_001.fastq F3D1_S189_L001_R2_001.fastq
Etc…

This file has to be saved as a MS DOS .txt file and then you have to delete the .txt extension in order for it to work. If you save it as a tab delimited .txt file, it won't work. It will appear as a wordpad icon.

5. Run: mothur > make.contigs(file=stability.files, processors=8)

What does it do?
make.contigs is a command that combines the forward and reverse reads of all the samples. It extracts the sequence and quality score data from the .fastq files, and joins the reads into a contig. Simply, it aligns the pairs of sequences and then looks across alignments to identify any bps where the two reads disagree. If one sequence has a base and the other has a gap, the quality score of the base must be over 25 to be considered real. If both sequences have a base at that position, then one of the bases has to have a quality score of 6 or more points better than the other. If one base isn't 6 points better than another, then the base is called as N.

Processors = 8 refers to the number of processors being used. If you're running this using your own laptop processors, you probably have 8, but if you are using Thor, you can use more processors. With Thor, usually 16 processors is ideal. This process takes time, and the more samples you have the longer it will take. For example, with 20 samples, this took about 1.5 minutes.

At the end of the process, mothur will give a list of each sample name and the number of full length sequences next to each name. It also creates several files that you need downstream. Stability.trim.contigs.fasta and stability.contigs.groups, which contain the sequence data and group identity for each sequence. Stability.contigs.report will tell you about the contig assembly for each read. Also mothur will give an error that says:

[WARNING]: your sequence names contained ':'. I changed them to '_' to avoid problems in your downstream analysis.

This is just telling you that mothur changed the sample names so that all "." are now "_", which will make things easier downstream.

Note - mothur keeps track of the number of processors you want to use and the files that have been created. If you ever want to check this, you can run the following command:

get.current()

6. Run: summary.seqs(fasta=stability.trim.contigs.fasta)

Example output of summary.seqs:

	Start	End	NBases	Ambigs	Polymer	NumSeqs	
Minimum:	1	248	248	0	3	1	
2.5%-tile:	1	252	252	0	3	3810	
25%-tile:	1	252	252	0	4	38091	
Median:	1	252	252	0	4	76181	
75%-tile:	1	253	253	0	5	114271	
97.5%-tile:	1	253	253	6	6	148552	
Maximum:	1	503	502	249	243	152360	
Mean:	1	252.811		252.811		0.697867	4.44854
# of Seqs:	152360						

This tells you the total number of sequences that are between 248 and 253 bases. It also tells you the length of the longest read and how many had ambiguous base calls. We want to get rid of reads that are too long or are ambiguous, which occurs in the next few steps.

7. Run: screen.seqs(fasta=stability.trim.contigs.fasta, group=stability.contigs.groups, maxambig=0, maxlength=275)

This is going to remove any sequences with ambiguous bases and that are longer than 275 bp. You could also run this a bit faster using:

screen.seqs(fasta=stability.trim.contigs.fasta, group=stability.contigs.groups, summary=stability.trim.contigs.summary, maxambig=0, maxlength=275)

This second option might be faster because the summary.seqs output file already has the number of ambiguous bases and sequence lengths calculated, so it's not necessary to recalculate them.

8. Run: unique.seqs(fasta=stability.trim.contigs.good.fasta)

What does this do?
unique.seqs is essentially a data reduction command. If 2 sequences have the identical sequence, they are considered duplicates and are merged. In the example dataset, after running unique.seqs, the amount of sequences was reduced from 129058 to 16477.

9. Run: count.seqs(name=stability.trim.contigs.good.names, group=stability.contigs.good.groups)

What does this do?
count.seqs generates a table where the rows are the names of the unique sequences and the columns are the names of the groups. The table is then filled in with the number of times each unique sequence shows up in each group.

10. Run: summary.seqs(count=stability.trim.contigs.good.count_table)

This takes a look at the number of sequences you are now dealing with and ensures that all sequences are the appropriate length and that there are 0 ambiguous sequences.

11. Run: pcr.seqs(fasta=silva.bacteria.fasta, start=11894, end=25319, keepdots=F, processors=8)

pcr.seqs will make a database that is customized to our region of interest. You need to have the reference alignment (silva.bacteria.fasta) in the folder along with your sequences and know where in that alignment your sequences start and end.

12. Run: system(rename silva.bacteria.pcr.fasta silva.v4.fasta)

This will rename your alignment to something useful.

13. Run: summary.seqs(fasta=silva.v4.fasta)

This will give you a summary of the alignment file.

14. Run: align.seqs(fasta=stability.trim.contigs.good.unique.fasta, reference=silva.v4.fasta)

This will align the sequences from your samples to the reference alignment. With the data reduction, it shouldn't take too long (it took 52 seconds with 20 samples).

15. Run: summary.seqs(fasta=stability.trim.contigs.good.unique.align, count=stability.trim.contigs.good.count_table)

    This will show you that there are some sequences that have insertions or deletions at the terminal ends of the alignments; also there are some sequences that start and end at the same position indicating poor alignment due to non-specific amplification.

16. Run: screen.seqs(fasta=stability.trim.contigs.good.unique.align, count=stability.trim.contigs.good.count_table, summary=stability.trim.contigs.good.unique.summary, start=1968, end=11550, maxhomop=8)

    Re-running screen.seqs will make sure that we're only working with things that overlap in the region that start at or before 1968 and end after position 11550. Also maxhomop will set the maxiumum homopolymer length to 8.

17. Run: summary.seqs(fasta=current, count=current)

This will now show that all sequences start and end at the same positions (1968 and 11550), but we want to make sure they only overlap that region.

18. Run: filter.seqs(fasta=stability.trim.contigs.good.unique.good.align, vertical=T, trump=.)

filter.seqs will remove overhangs at both ends of the alignment – this shouldn't be much of an issue with paired-end sequencing, but could be with single direction sequencing. Additionally, it will remove columns in the alignment that only contain gap characters.

At the end of running filter.seqs, you get some information:

Length of filtered alignment: 376
Number of columns removed: 13049
Length of the original alignment: 13425
Number of sequences used to construct filter: 16349

This tells you that the initial alignment was 13425 columns wide and that we were able to remove 13049 terminal gap characters using trump=/ and vertical gap characters using vertical=T. The final alignment is only 376 columns. By removing these gaps, we've probably created some redundancy across sequences by trimming the ends, so we can run unique. seqs again to make sure that there are only unique sequences in the alignment.

19. Run: unique.seqs(fasta=stability.trim.contigs.good.unique.good.filter.fasta, count=stability.trim.contigs.good.good.count_table)

    Running unique.seqs identified 3 duplicate sequences, so now there are 16345 instead of 16348 unique sequences.

20. Run: pre.cluster(fasta=stability.trim.contigs.good.unique.good.filter.unique.fasta, count=stability.trim.contigs.good.unique.good.filter.count_table, diffs=2)

pre.cluster is going to further denoise sequences by allowing for up to 2 differences between sequences to group them together. This command splits the sequences by group and then sorts them by abundance from most abundant to least abundant and then identifies sequences that are within 2 nucleotides of each other and merges them. The general rule is that there is 1 difference allowed for every 100 bp of sequences.

21. Run: summary.seqs(fasta=current, count=current)

Running summary.seqs again will tell you that you now have 5661 unique sequences.

22. You need to download have a separate executable file called uchime.exe in the folder with all your data to run the next command. You likely downloaded it with mothur.exe, but you might get an error if the uchime.exe file isn't in that folder.

23. Run: chimera.uchime(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.count_table, dereplicate=t)

UCHIME is an algorithm that removes chimeras. This command splits the data by sample and checks for chimeras using the most abundant sequences as a reference. If a sequences is flagged as a chimera in

one sample, the default (dereplicate=F) is to remove it from all samples. This is a bit aggressive, since sometimes rare sequences get flagged as chimeric when they're the most abundant sequence in another sample.

24. Run: remove.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.fasta, accnos=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.accnos)

This will remove chimeric sequences from the fasta file. Uchime only removes them from the count file.

25. Run: summary.seqs(fasta=current, count=current)

This shows you that the fasta file is now down to 2612 sequences after removing chimeras and that about 7% of the sequences were flagged as chimeric. This is reasonable – if you had a lot more that this you could question the sequencing facility or your library prep.

26. Run: classify.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count_table, reference=trainset9_032012.pds.fasta, taxonomy=trainset9_032012.pds.tax, cutoff=80)

classify.seqs is the first step toward identifying your sequences and will classify them as Bacterial 16S rRNA vs. other things, such as 18S, 16S rRNA in Archaea, chloroplasts and mitochondria. This could all depend on the primers you're using, but if you used primers that are just supposed to amplify Bacteria, then they're making a mistake by amplifying Eukarotes and Archaea.

27. Run:
remove.lineage(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count_table, taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.taxonomy, taxon=Chloroplast-Mitochondria-unknown-Archaea-Eukaryota)

remove.lineage is going to remove all the unwanted lineages from your analysis (in this case, taxon = chloroplast mitochondria, Archaea and Eukaryota). You could leave Archaea in there if you wanted to, but since this particular primer set was only meant to amplify Bacteria, you should take them out. It does not remove unknown sequences.

23. Run: summary.seqs(fasta=current, count=current)

This tells us that there are now 2592 unique sequences and a total of 119301 sequences. So, about 350 of the sequences were within the various undesireable groups. This is the end of sequence curation, so now you can see what the error rate is.

Optional – only if you ran a mock community sample!

24. Run:
get.groups(count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.count_table, fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.fasta, groups=Mock)

The next few commands rely upon the mock community data. If you didn't sequence a control mock community, then this won't work. In the Schloss lab training dataset, it tells us that we had 68 unique sequences and a total of 4061 total sequences in the Mock community.

24. Run:

seq.error(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, reference=HMP_MOCK.v35.fasta, aligned=F)

They haven't quite ironed out how to calculate the sequence error rate in mothur, but have instructions on how to do it in R. The error rate is 0.0065%, so that's good.

25. Run the following 4 commands:

dist.seqs(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, cutoff=0.20)

cluster(column=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table)

make.shared(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.list, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table, label=0.03)

rarefaction.single(shared=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.shared)

These commands will produce a file called stability.trim.contigs.good.unique.filter.unique.precluster.pick.pick.pick.an.unique_list.groups.rarefaction. When you open it you'll see that for 4061 sequences, we'd have 35 OTUs from the mock community. This indicates that there are some chimeras that escaped detection with uchime. With no errors or chimeras, we're expecting 20 OTUs, so it's not perfect, but it's pretty good.

26. Run:

remove.groups(count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.count_table, fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.fasta, taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.taxonomy, groups=Mock)

Now it's time to start analyzing the actual data. First you need to get rid of the Mock sample from the dataset using the remove.groups command.

28. Option 1 – Run the following 2 commands:

dist.seqs(fasta= stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, cutoff=0.20)

cluster(column=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.dist, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table)

Option 2 – Run:

cluster.split(fasta=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table, taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy, splitmethod=classify, taxlevel=4, cutoff=0.15)

dist.seqs and cluster works well for a small dataset. Alternatively the cluster.split command uses taxonomic information to split the sequences into bins and then cluster within each bin. If you split at the level of Order/Family, and cluster to a 0.03 cutoff, you'll get just as good clustering as you would with the traditional approach. The advantage of cluster.split is that it is faster, uses less memory and can be run on multiple processors. In this command taxlevel=4 corresponds to the Order level. You could change that if you only wanted Phylum. (And actually, when I ran dist.seqs + cluster, it crashed mothur, but when I ran cluster.split it worked just fine, so I guess that's the better way to go.)

29. Run:
    make.shared(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.list, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table, label=0.03)

We need to know how many sequences are in each OTU from each group, which is done using make.shared. This command indicates that we're only interested in a 0.03 cutoff level.

30. Run:
    classify.otu(list=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.list, count=stability.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count_table, taxonomy=stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy, label=0.03)

This command will tell us the taxonomy for each OTU. If you open the file called stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.0.03.cons.taxonomy, you'll see a list of OTU name, the size (or amount of sequences in that OTU) and the taxonomy. But it doesn't tell you which OTUs are from which sample.

OTU-Based Analysis

31. Run:

system(rename stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.shared stability.an.shared)
system(rename stability.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.an.unique_list.0.03.cons.taxonomy stability.an.cons.taxonomy)

These commands will just rename the shared and consensus taxonomy files.

32. Run: count.groups(shared=stability.an.shared)

This will tell us how many sequences there are in each sample. The smallest sample had 2441 sequences in it.

33. Run: sub.sample(shared=stability.an.shared, size=2441)

This will subsample the data so that each sample has 2441 sequences per sample. This command makes a file that tells you which OTUs are in each sample and it's called: stability.an.0.03.subsample.shared. You could basically stop here and use this file to make NMDS or PCoA or whatever you want in R, or you can continue with mothur.

34. Run: collect.single(shared=stability.an.shared, calc=chao-invsimpson, freq=100)

First thing to do is calculate alpha diversity of each sample. This step generates a collector's curve of the Chao1 richness estimators and the inverse Simpson diversity index. The files called .chao and .invsimpson can be plotted to yield a collectors curve. The Chao1 curve continues to climb with sampling, but the inverse Simpson diversity indices are relatively stable. It's possible to compare the richness of these communities using the inverse Simpson index. Chao1 is really a measure of the minimum richness in the community, not the full richness in the community.

35. Run: rarefaction.single(shared=stability.an.shared, calc=sobs, freq=100)

    This will generate a .rarefaction file, which can be plotted. Rarefaction is a measure of diversity. If you consider two communities with the same species richness, but different evenness, then after sampling a large number of individuals their rarefaction curves with asymptote to the same value, but since they have different evennesses, the curves will differ. So, rarefaction only allows for comparisons between samples based on diversity and not richness.

36. Run: summary.single(shared=stability.an.shared, calc=nseqs-coverage-sobs-invsimpson, subsample=2441)

This will output the data into a table file called stability.an.groups.ave-std.summary. This is the table you really want, because it has the number of sequences, the sample coverage and the number of observed OTUs, the inverse simpson diversity estimate. 2441 sequences from each sample were randomly sampled 1000 times to calculate the average.

Beta Diversity

37. Run: heatmap.bin(shared=stability.an.0.03.subsample.shared, scale=log2, numotu=50)

This will make a heatmap of the relative abundance of each OTU across the 24 samples using a heatmap. It only uses the top 50 OTUs.