

MOTHUR Tutorial for Fungal Community Analysis

INTRODUCTION

www.mothur.org

Before we get started, there are some things you should know...

MOTHUR works with a command line interface. This may sound intimidating, but if you take a few hours to learn some basic operations in your Unix/Linux/Terminal, it will save you *LOTS* of time and frustration in the future. There are plenty of great free online tutorials to help you out. Here's the first of ~6,000,000 google hits for "unix tutorial":

<http://www.ee.surrey.ac.uk/Teaching/Unix/>

Eukaryotic systems in general, and ITS analyses specifically, are a little outside of the microbial ecology mainstream. MOTHUR was designed to work with environmental sequences that can be reasonably aligned **and** are also useful for distinguishing species. In prokaryotic systems the 16s (a.k.a. the nuclear gene coding the small subunit of the ribosome) seems to fit both of these criteria. In fungi we're not so lucky. The ITS region is good at distinguishing species, but is difficult to get into a meaningful alignment across kingdom Fungi. Other loci, like the large or small ribosomal subunits (28s or 18s) can yield reasonable multiple sequence alignments, but might not be good for determining **Operational Taxonomic Units (OTUs)** depending on which groups of fungi you're interested in.

To make a long story short, the parts of MOTHUR that deal with clustering OTUs based on multiple sequence alignments probably won't work for you no matter which locus you're working with. That's OK! There are a number of programs which use pairwise alignments and will work for you just fine: (i.e. Blastclust, CD-HIT, UCLUST, TIGCL) and at this meeting we'll learn about some analysis pipelines which integrate these types of clustering algorithms to enable you to work with programs like MOTHUR. By the time you read this, this may even be a part of MOTHUR.

This tutorial is divided into two sections: a "**Phylogentic Analysis**" in which we use MOTHUR on our raw 454 data of environmental LSU sequences, and an "**OTU Analysis**" section in which I've already used the program CD-HIT-EST to calculate OTUs based on 97% ITS sequence identity. The data used in this tutorial is from fungi found in indoor dust samples from a global sample (see Amend A, Samson R, Seifert K, Bruns T (2010) Deep sequencing reveals diverse and geographically structured assemblages of fungi in indoor dust. *Proceedings of the National Academy of Sciences*, doi:10.1073/pnas.1000454107 for details). We've reduced the number of reads to make things simpler.

Finally, MOTHUR can do a lot more than I had time to present in this tutorial, and it is constantly being improved upon and expanded. If this is a

program that you're planning to use, I urge you to consult the MOTHUR manual for alternative analysis options and parameters (sections which are indented below were "borrowed" from this source).

STARTING MOTHUR

In your terminal, navigate to your "FesinFiles" folder. On my machine I type:

```
crassa$ cd /Users/crassa/Documents/Fesin/FesinFiles
```

My computer is named "crassa" and the "\$" indicates the UNIX prompt, so I don't actually type those. The UNIX command "cd" stands for "change directory". You can save yourself the trouble of typing the entire path by locating the file in your "finder" or windows directory and dragging it into your terminal. The path, as if by magic, appears.

Now that we're inside the right directory, the easiest way to execute MOTHUR is by simply indicating the path where you have the MOTHUR executable installed. Again it's pretty easy to do this by finding the file and dragging it in:

```
FesinFiles crassa$ /Applications/mothur
```

OK, we're ready to go! MOTHUR will always look for input files from, and will output files to the directory from which it is executed (FesinFiles in this case) so if the program can't find files that you KNOW are there, you should make sure that you are working from the correct directory.

PHYLOGENETIC (LSU) ANALYSES

There are three files we'll be using here LSU.fna , LSU.qual, and LSUoligos. *.fna is equivalent to a fasta sequence file, and *.qual is a quality score file. These are typical outputs from a 454 run.

Initial processing and clean-up of sequences

Let's take a quick look at the LSU.fna file using "Summary.seq"

```
MOTHUR > summary.seqs(fasta=LSU.fna)
```

	Start	End	NBases	Ambigs	Polymer
Minimum:	1	55	55	0	3
2.5%-tile:	1	104	104	0	4
25%-tile:	1	278	278	0	4
Median:	1	345	345	0	5
75%-tile:	1	392	392	0	5

97.5%-tile:	1	484	484	1	6
Maximum:	1	572	572	5	61
# of Seqs:	1936				

So we can see that we have 1936 sequences, that our shortest read was 55 bp, our longest was 572 bp., and the median was 345 bp. We can also see that our “messiest” sequence contained 5 ambiguous base calls (Ns), and there was at least one sequence with a homopolymer (single nucleotide repeat) 61 bases long!

Let’s clean these sequences up a little bit. Here’s what Pat Schloss says about the Trim.seqs command:

The **trim.seqs** command provides the preprocessing features needed to screen and sort pyrosequences. Similar analyses are provided by [RDP](#); here we give you added flexibility and speed. The command will enable you to trim off primer sequences and barcodes, use the barcode information to generate a group file and split a fasta file into sub-files, screen sequences based on the qual file that comes from 454 sequencers, cull sequences based on sequence length and the presence of ambiguous bases and get the reverse complement of your sequences. While this analysis is clearly geared towards pyrosequencing collections, it can also be used with traditional Sanger sequences.

We used “barcoded” primers to multiplex a number of samples in a single run. To be able to figure out which sequences go with which samples, we created an “oligos” file:

The oligos option takes a file that can contain the sequences of the forward and reverse primers and barcodes and their sample identifier. Each line of the oligos file can start with the key words "forward", "reverse", and "barcode" or it can start with a "#" to tell MOTHUR to ignore that line of the oligos file. Here we are using a "#" for the reverse primer to indicate that we don't want MOTHUR to screen for the reverse primer because the 454 sequencing platform cannot generate sequences that are >500 bp long. If the "#" were removed, all of the sequences would wind up in the scrap file. You can enter your oligos as upper or lowercase letters. It has been shown that sequencing errors in the PCR primer region of a sequence correlate highly with poor sequence quality. Therefore, MOTHUR will not allow you to have anything less than an exact match to the primer or barcode sequences that you provide.

So let’s have MOTHUR sort our sequences by barcode:

```
MOTHUR > trim.seqs(fasta=LSU.fna, oligos=LSUoligos)
```

You'll notice four new files in your directory:
 LSU.scrap.fasta-the sequences that didn't pass
 LSU.trim.fasta-the sequences that did pass
 LSU.trim.fasta.summary-a sequence by sequence analysis of what failed/passed
 And
 LSU.groups-all of our sequences sorted by barcode into the samples from which they came

And then let's check our results:

```
MOTHUR > summary.seqs(fasta=LSU.trim.fasta)
```

	Start	End	NBases	Ambigs	Polymer
Minimum:	1	24	24	0	3
2.5%-tile:	1	73	73	0	4
25%-tile:	1	247	247	0	4
Median:	1	314	314	0	5
75%-tile:	1	360	360	0	5
97.5%-tile:	1	450	450	1	6
Maximum:	1	521	521	4	14
# of Seqs:	1782				

So it looks like only 154 sequences were "scrapped" including the sequence with a 61 bp homopolymer. You'll also notice that our shortest sequences are now 31 bp shorter. That's because MOTHUR removed the priming region and 8 bp barcode from our sequences.

We can also trim our sequences based on quality scores.

In addition to a .fna file, the 454 platform generates a .qual file which mirrors the fna file. The difference is that in place of letters, the qual file has numbers indicating the quality of each base. For example, an "N" will be represented by a "0" because it is a poor base call. If a qfile is provided, you must provide either the qaverage or qthreshold options as well.

MOTHUR has a number of other quality-checking options:

- Qaverage-average Q score of a sequence**
- Qthreshold-threshold Q score of a sequence**
- Qtrim-trim away low quality sections of a sequence**
- Maxambig-the number of allowable Ns**
- Maxhomop-the length of tolerable homopolymers**
- Minlength-the minimum length of sequences**
- Maxlength-the maximum length of sequences**

These can be combined in a single command and changed as you see fit. Here's a moderately tolerant quality check:

```
MOTHUR > trim.seqs(fasta=LSU.fna, qfile=LSU.qual, oligos=LSUoligos,  
minlength=300, maxlength=500, maxambig=0, maxhomop=10, qaverage=25)
```

Check this file with

```
MOTHUR > summary.seqs(fasta=LSU.trim.fasta)
```

As you can see, ~50% of our original sequences passed.

Unique Sequences

You may be working in a system with relatively low complexity, or your samples may be dominated by a few species. To save hard drive space and computational effort, MOTHR will collapse all of the identical sequences in a “.names” file and will generate a list of unique sequences.

```
MOTHUR > unique.seqs(fasta=LSU.trim.fasta)
```

This command works on “identical” sequences in the strict sense, meaning redundant sequences must have exact nucleotide matches and sequence length. In this case this didn’t help reduce our file size, but the “.names” file will still be useful later on.

Align sequences

MOTHR has a built-in aligner that, in my opinion has its advantages and disadvantages. It can handle ENORMOUS amounts of data and process them relatively rapidly, and it produces alignments that are, obviously, easy to use in subsequent MOTHR analyses (like chimera checking). However, it requires a “seed” or “template” alignment off which to work. Depending on which locus you are working with, you may have to find or create your own such template alignment.

I’ve taken the sequences from Tim James’ ncLSU alignment off of the AFTOL website, removed the non-fungal taxa, re-aligned with MAFFT, and then removed the empty columns or the positions downstream from our target locus. This is the “james.trim.mafft” file. We can use this as our “template alignment”.

To align our sequences based on this template alignment we type:

```
MOTHUR> align.seqs(candidate=LSU.trim.unique.fasta,  
template=james.trim.mafft, flip=T)
```

The flip=T flag will take care of any reverse complimentary sequences in our alignments if found.

Check for Chimeras

This works by comparing the closest match of the first half of our sequences to the template alignment and the closest match of the other half of our sequences to the template alignment. In theory, if there are two different matches, our sequence may be chimeric. Of course this is not always the case, and it will probably always generate some false positives. Depending on what type of analysis you're doing, this may or may not be alright. If your dataset is small enough, it's always best to check these by hand and decide for yourself. There are **many** parameters for checking chimeras in MOTHUR, let's use the default settings for now.

```
MOTHUR> chimera.slayer(fasta=LSU.trim.unique.align,  
template=james.trim.mafft)
```

You can tell from the output on the screen that the chimera program found 4 (putative) chimeras. Let's remove those from our analyses using the "remove.seqs" function. We should make sure that we remove those sequences from our groups, name **AND** align file so that everything matches up.

```
MOTHUR> remove.seqs(accnos=LSU.trim.unique.slayer.accnos,  
name=LSU.trim.names)
```

```
MOTHUR> remove.seqs(accnos=LSU.trim.unique.slayer.accnos,  
group=LSU.groups)
```

```
MOTHUR> remove.seqs(accnos=LSU.trim.unique.slayer.accnos,  
fasta=LSU.trim.unique.align)
```

Let's take a "look" at our alignment using the "summary.seqs" command:

```
MOTHUR > summary.seqs(fasta=LSU.trim.unique.align)
```

	Start	End	NBases	Ambigs	Polymer
Minimum:	113	541	271	0	4
2.5%-tile:	163	549	294	0	4
25%-tile:	164	574	322	0	4
Median:	164	602	346	0	5
75%-tile:	164	641	382	0	5
97.5%-tile:	191	803	455	0	6
Maximum:	209	890	496	0	7
# of Seqs:	868				

You'll see that we have 4 fewer sequences now that the chimeras have been removed. Everything else appears to be reasonable, except that our alignment seems to start at position 113. It must contain a lot of "gaps". Let's trim that down using the "filter.seqs" command to get rid of all of our alignment positions (columns) that only contain gap characters.

```
MOTHUR > filter.seqs(fasta=LSU.trim.unique.align)
```



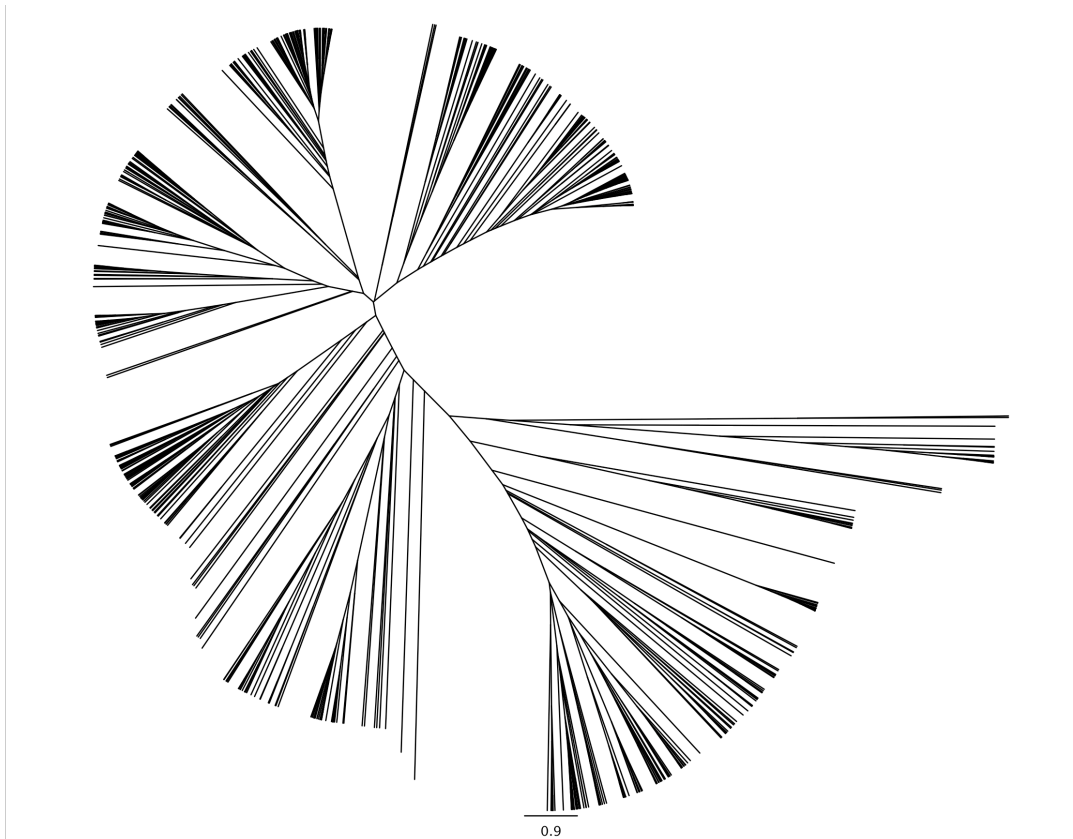
now check the new alignment:

```
MOTHUR > summary.seqs(fasta=LSU.trim.unique.filter.fasta)
```

As you can see, our alignment size is reduced by about 70%, that should help us save some memory later. Thanks MOTHUR!

Produce a phylogenetic tree

Here's a quick distance tree that I produced using the program JALVIEW. It's in your folder as "LSU.distance.tre." Some people like to build trees for large datasets using the program ClearCut, and a wrapper is included in MOTHUR for this program. I prefer to use FastTree, but the trees that this program produces aren't easily compatible with MOTHUR.



Compare community phylogenetic structure

MOTHUR offers 3 different types of analysis: Parsimony, Unifrac, and Libshuff. Each offers variations on how to test if the phylogenetic community structure in one sample differs from the others.

Parsimony test

This test generates random trees from a pool of “species” and compares the number of changes in tree topology accounting for differences among communities against the randomized null. Let’s first start by reading the tree into memory along with the group and name file:

```
MOTHUR > read.tree(tree=LSU.distance.tre, group=LSU.pick.groups,  
name=LSU.trim.pick.names)
```

Now let’s test if ANY of the samples differ from random structure. In the interest of time, let’s reduce the number of randomizations from 1,000 (default) to 100.

```
MOTHUR> parsimony(iters=100)
```

We get a significance score <0.01 which tells us that the community structure of at least one sample differs from random. Let's take a look at the pairwise comparisons of these samples. This can be very computationally intensive (and increases exponentially with the number of samples), so just to demonstrate how this works let's tell MOTHUR to compute 100 randomizations. For a real study you'd probably want more iterations.

```
MOTHUR> parsimony(groups=all, iters=100)
```

If you look at your ".psummary" file you'll see all of the pairwise comparisons. It appears here that all samples differ significantly from each other. The statistic, in this case, is just a measure of significance and shouldn't be interpreted as a degree of dissimilarity.

UniFrac

This is another metric of community phylogenetic dissimilarity. Take a look at the UniFrac website for more information: bmf2.colorado.edu/unifrac/index.psp There are two "versions" of this program: a weighted and an unweighted version. In the weighted version, the abundance of sequences is taken into account along with phylogenetic similarity (shared branch length) to calculate how similar communities are. In the unweighted version, sequence abundance is not taken into account.

Let's try the unweighted version, again reducing the number of iterations in the interest of saving time:

```
MOTHUR > unifrac.unweighted(iters=100)
```

The p-value here indicates that at least one of the samples here differs from randomly generated trees at $p<0.01$

Let's compare our groups to each other and generate a distance matrix:

```
MOTHUR > unifrac.unweighted(groups=all, iters=100, distance=true)
```

Our output will give us the pairwise differences between communities, and the ".dist" file gives us this information in a distance matrix that we can use for ordinations or multivariate statistics. There is a built-in function to generate a Principal Components (PCoA) ordination using the distance matrices.

Type:



```
MOTHUR> pcoa(dist=LSU.distance.tre.unweighted.dist)
```

This will generate the files to produce a scatter plot (using the first two or three axes) and a “loadings” file to determine how much variance was described by each axis.

OTU (ITS) ANALYSES

For this part of the tutorial we'll skip the parts where we clean up and sort our sequences, and assume that we have already generated our ".list" and ".group" files. A list file contains a list of all the reads that have been clustered into each OTU, and a group file defines from where each read originated (Australia, Canada and Micronesia here). I generated these files using a program called CD-HIT-EST (www.bioinformatics.org/cd-hit) and then some Unix code, but there are various ways to cluster your sequences into OTUs, some of which have been presented in this workshop. By the time you read this, a pairwise aligner/OTU clusterer may have been added to MOTHUR.

Input your files into MOTHUR

```
MOTHUR > read.otu(list=Fesin100.list, group=Fesin100.group)
```

This will generate three new types of files: a) a log file to keep track of your commands, b) a *.rabund file which is the frequency distribution of OTUs in each sample and c) a Fesin100.shared file.

Open the Fesin100.shared file with Excel and take a look.

0.03	AUS1	237	0	0	0	0	2
0.03	AUS3	237	0	0	0	0	0
0.03	AUS5	237	0	3	1	0	1
0.03	AUS6	237	0	0	1	3	0
0.03	CAN1	237	0	0	0	0	1
0.03	CAN12	237	0	0	0	0	0
0.03	CAN29	237	3	0	0	0	4
0.03	MICRO1	237	0	0	0	0	0
0.03	MICRO2	237	0	0	0	0	0
0.03	MICRO3	237	0	0	0	0	0
							...

This is a sample by species matrix, which is very useful for other statistics programs like *R*, or *CANOCO*. Each row is one of the samples we note in our "groups" file. The first column is our % Identity cut-off, the second column is the name of our sample. The third column is the number of OTUs in our study. The next column (and all of the rest) represents an OTU. The first OTU was found only in one sample CAN29, and it was found in 3 reads. The second OTU was found three times in sample AUS5 etc...

The OTUs are in the same order as our list file, so with a little copy/paste you could put headings on these columns. You can also use the command

```
MOTHUR > get.otulist(list=Fesin100.list)
```

To generate a file “Fesin100.0.03.otu” which is a list of the sequences associated with each OTU in order:

```
1    FX2FH6V04EM0ED, FX2FH6V04EXJPC, FX2FH6V04D3UW1
2    FX2FH6V02CG101, FX2FH6V02CMEMX, FX2FH6V02B2IFL
3    FX2FH6V02B1VQ5, FX2FH6V02BU96P
4    FX2FH6V02B68CA, FX2FH6V02B5GZN, FX2FH6V02CERN1
5    FX2FH6V02BYJVU, FX2FH6V01BBEUK, FX2FH6V01AURZ8,
    FX2FH6V04D895P, FX2FH6V04EWS5W, FX2FH6V04ENN60, FX2FH6V04
    ESJ3K, FX2FH6V04EAOVN
```

...

MOTHUR has a built in function to visualize the shared file as a heatmap:

```
MOTHUR> read.otu(list=Fesin100.list, group=Fesin100.group)
MOTHUR> heatmap.bin(sorted=T)
```

The presence of each OTU is noted by rectangle, and the “abundance” of that sequence is color-coded. We’ve elected to sort the OTU’s by the order in our “.list” file. If we remove the “sorted=T” command above, MOTHUR will sort the OTUs from most common to least common.

Calculate Diversity Estimates

MOTHUR can calculate the diversity of your samples using any of 3 methods:

- Collector’s curves
- α diversity statistics
- rarefaction curves

The commands for all of these are essentially the same, and there are various parameters that can be changed (such as the number of randomizations) which I won’t cover here.

If you want to study the diversity of all of your samples as a pool, just read in the “list” file:

```
MOTHUR > read.otu(list=Fesin100.list)
```

Now type:

```
MOTHUR > collect.single()
```

If you go to your folder, you’ll see 8 new files: Fesin100.sobs, Fesin100.simpsons etc. Each of these contains XY data for a different diversity calculator. For example open Fesin100.chao:

#sampled	0.03	LCI	HCI
1	1	0	0
100	209.5	119.5565	433.0348
200	239.2727	159.7036	408.3879
300	298.0667	210.4678	468.6519
400	402.1176	284.4829	618.9611
500	319	249.3621	441.7171
600	332.5	270.5085	437.4255
700	358.0857	291.7361	469.0217
800	376.75	311.9079	482.85
900	408	338.1363	521.0059
1000	404.5532	341.8811	504.6753

In this file the first column tells you how many sequences have been sampled; this would typically be plotted on the x-axis of a graph. The data in the second column tells you how many OTUs were observed for that label and number of sequences. By default, `collect.single()` prints output every 100 sequences. For some calculators there are formulae that enable us to determine the 95% confidence intervals. The following two columns contain the bounds on the 95% confidence interval.

If you're just interested in the statistics, and not plotting a graph, you can enter

```
MOTHUR > summary.single()
```

This will generate the summary statistics for all of these calculators in a single file.

Finally you can use MOTHR to generate a rarefaction curve:

```
MOTHUR > rarefaction.single()
```

You may be interested in calculating each of these diversity estimates on each sample individually, not just for all of them combined. To do this read your "group" file into memory (and re-read the "list" file)

```
MOTHUR > read.otu(list=Fesin100.list, group=Fesin100.group)
```

The commands are exactly the same, except you will now get a separate file for each sample! This can quickly overload your computer, so it's a good idea to limit the output to **only** the tests that you're interested in. You can do this using the "calc" flag. For example:

```
MOTHUR > collect.single(calc=sobs-chao)
```

This will only output files for the sobs and chao calculations. Because the default is to output every 100 sequences, we only get two data points. Not much of a curve! Let's have MOTHUR print the output every 5 sequences instead:

```
MOTHUR > read.otu(list=Fesin100.list, group=Fesin100.group)
```

```
MOTHUR > collect.single(calc=sobs-chao, freq=5)
```

numsampled	0.03	Lci	hci
1	1	0	0
5	5.5	4.1492	19.0773
10	29	13.1544	93.5578
15	17	11.341	46.5405
20	21.3333	13.9371	56.9707
25	26.25	17.6522	62.7206

Changing the "freq" flag is also useful if you're processing large numbers of samples since you can output every 1,000 or 10,000 sequences (instead of 100) to reduce the size of your files.

Beta-diversity Statistics

Most of the time you will probably be interested in comparing samples against each other. MOTHUR gives us several ways of doing this based on how many OTUs are shared among samples. Let's start with something easy.

```
MOTHUR > read.otu(list=Fesin100.list, group=Fesin100.group)
```

```
MOTHUR > tree.shared()
```

This will output two Newick formatted tree files (.tre) using the Jaccard coefficient and the Yue & Clayton theta. You can select alternative calculators using the "calc" function (i.e. MOTHUR > tree.shared(calc=sorest)). MOTHUR includes the following calculators (at the time of writing):

Similarity in community membership (Presence/Absence)

[anderberg](#) - the Anderberg similarity coefficient

[jclass](#) - the traditional Jaccard similarity coefficient based on the observed richness

[jest](#) - the Jaccard similarity coefficient based on the Chao1 estimated richnesses

[kulczynski](#) - the Kulczynski similarity coefficient

[kulczynskicody](#) - the Kulczynski-Cody similarity coefficient

[lennon](#) - the Lennon similarity coefficient

[ochiai](#) - the Ochiai similarity coefficient

[sorclass](#) - the Sorenson similarity coefficient based on the observed richness

sorest - the Sorenson similarity coefficient based on the Chao1 estimated richnesses

whittaker - the Whittaker similarity coefficient

Similarity in community structure (Abundance-Based)

braycurtis - the Bray-Curtis similarity coefficient

jabund - the abundance-based Jaccard similarity coefficient

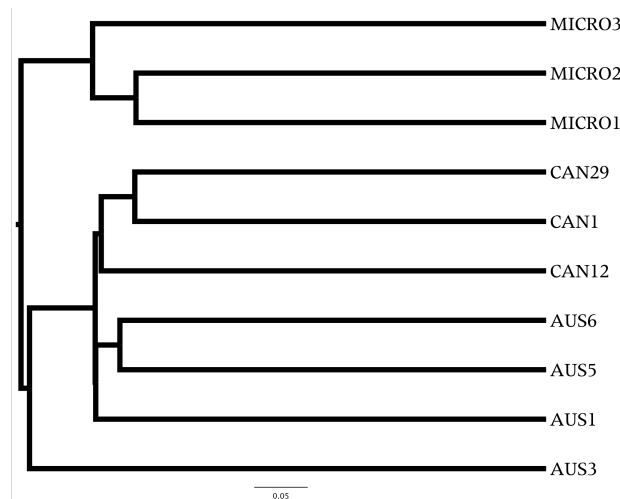
morisitahorn - the Morisita-Horn similarity coefficient

sorabund - the abundance-based Sorenson similarity coefficient

thetan - the Smith theta similarity coefficient

thetayc - the Yue & Clayton theta similarity coefficient

If you open these files using a tree viewing program like “treeview” the webbased “ItoI” or, my favorite “figtree” you can visualize how dis-similar your samples are from each other in a dendrogram like this:



So it looks like the samples group together by geography, with AUS3 as a bit of an outgroup. Another way to visualize these relationships with the same dissimilarity metrics would be in a heatmap. Try:

```
MOTHUR > heatmap.sim()
```

The output displays how similar two samples are by the intensity or color of a square. It doesn't look great here, but in other circumstances it could be useful.

MOTHUR will also produce Venn diagrams (showing the number of overlapping OTU) of up to 4 groups, let's choose samples using the “groups” flag:

```
MOTHUR > read.otu(list=Fesin100.list, group=Fesin100.group)  
MOTHUR > venn(groups=AUS3-MICRO3-MICRO2-CAN1)
```

