Lemmon Data Workflow

Ian G. Brennan

Contents

Install the software/code:

1. ASTRID (v1.4)[1]: http://pranjalv123.github.io/ASTRID/
   - chuck the executable from (https://github.com/pranjalv123/ASTRID/releases) into the folder
2. ASTRAL (4.10.8)[2]: https://github.com/smirarab/ASTRAL (get the zip from "installation")
3. R (v3.3.1): https://cran.rstudio.com/
4. R Studio (v0.99.903): https://www.rstudio.com/products/rstudio/download2/
5. FigTree (v1.4.3): https://github.com/rambaut/figtree/releases/tag/1.4.3pre
6. BEAST 2 (v2.4.2)[3]: www.beast2.org
   - this will include Tracer, TreeAnnotator, and DensiTree
7. Canopy: https://store.enthought.com/downloads/#default
   - create an academic account, and download the better, free version
8. EAPhy[4]: https://github.com/MozesBlom/EAPhy
   - download the whole folder using the "Clone or download" option on the right
9. Galaxy[5]: https://getgalaxy.org
   - register an account, download and install according to directions below
10. Blast+ [6]: https://github.com/peterjc/galaxy-blast
    - pick it up from the galaxy toolshed by searching for 'blast' or 'blast_plus'

---

[1]Vachaspati, P. & Warnow, T. ASTRID: Accurate Species TRees from Internode Distances. BMC Genomics 16, 1-13, doi:10.1186/1471-2164-16-s10-s3 (2015).

[2]Mirarab, S. & Warnow, T. ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. Bioinformatics 31, doi:10.1093/bioinformatics/btv234 (2015).

[3]Bouckaert, R. et al. BEAST 2: A Software Platform for Bayesian Evolutionary Analysis. PLoS Comput Biol 10, e1003537, doi:10.1371/journal.pcbi.1003537 (2014).

[4]Blom, M. P. K. EAPhy: A Flexible Tool for High-throughput Quality Filtering of Exon-alignments and Data Processing for Phylogenetic Methods. PLoS Currents 7, ecurrents.tol.75134257bd75134389c75134204bc75134251d75134226d75134242aa75139089f, doi:10.1371/currents.tol.75134257bd389c04bc1d26d42aa9089f (2015).

[5]Afgan, E. et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses. Nucleic Acids Research, 44(W1): W3-W10, doi:10.1093/nar/gkw343 (2016).

[6]Cock, P.J.A, Chilton, J.M., Gruening, B., Johnson, J.E., Soranzo, N. NCBI BLAST+ integrated into Galaxy. GigaScience, 4:39, doi:http://dx.doi.org/10.1186/s13742-015-0080-7 (2015).

Species Trees Using the Multi-Species Coalescent

Ten seconds of theory:

When sampling hundreds of loci, you're inevitably going to come across some that are doing funny things (i.e. divergent topologies), this a result of a number of differing biological processes (incomplete lineage sorting, hybridization, etc.). We want to limit the effects of these loci, so we'll address it in three ways (a) mechanically: remove loci contributing little to the overall phylogenetic resolution; (b) brut-force: swamp divergent signal by providing hundreds of other loci which give the correct signal; (c) theoretically: use the coalescent to limit the effects of incomplete lineage sorting (ILS). Methods (a) and (b) will be used later on, but we'll start with the more elegant way (c) by using the multi-species coalescent (MSC—thorough) and distance-based methods (very fast) in ASTRAL and ASTRID, respectively, to outperform concatenation in a likelihood framework (RAxML). One last thing: our distance-based tree estimation methods (ASTRAL/ASTRID) rely on gene trees. We'll get those from individual RAxML runs (and bootstraps) completed on each locus.

Get ASTRID up and running:

1. open up your terminal, navigate to your ASTRID folder
    - "ls" (list the folders), then "cd" (change directory) to the proper folder
2. check for compatibility by typing in

```
$ ./ASTRID-osx
```

```
+ or check usage here: https://github.com/pranjalv123/ASTRID
```

The quickest/dirtiest tree you'll make goes something like this:

1. in the ASTRAL folder, find the RAxML_bestTree_ALL.txt this holds the best tree per locus from the RAxML analysis. This will be our ASTRID input file.
2. open up your terminal, navigate to your ASTRID folder, and we'll run the ASTRID analysis explicitly from our input of [your number here] gene trees:

```
$ ./ASTRID -i [drag/drop input file: "RAxML_bestTree_ALL.txt"]
```

3. additionally, we didn't even save the tree, so let's rerun it with the output designated by "-o". This will get saved to the same folder the ASTRID executable is in.

```
$ ./ASTRID -i [drag/drop input file: "RAxML_bestTree_ALL.txt"] -o output.tre
```

4. check it out in figtree, notice that there aren't any support values? That's because we didn't bootstrap this at all. Additionally, the shortcoming here is that this is explicitly estimating the species tree from our gene trees without taking into consideration any sort of support/confidence metric from each of our gene trees, meaning they're all perfect and equal. We know this isn't true, so let's take into account the bootstraps for each gene tree.

Now let's run the species tree using the BS values from RAxML:

1. find the bootstrap files. There should be 1 per locus. Each one of these files contains the 100 bootstrap replicates from the RAxML analysis, which will inform our ASTRID run. They'll be located in the RAxML_2alleles folder.
2. your terminal should already be open and directed to the ASTRID folder, so I'll stop saying this.
3. run the ASTRID analysis including a list of where the BS files are:

```
$ ./ASTRID -i [drag/drop input file: "RAxML_bestTree_ALL.txt"] --bslist [drag/drop all your BS files he
```

4. have a look this time, it should be much closer to the true species tree. This has taken into account the relative confidence of each locus, and bootstrapped the tree itself (100x), so each node should have a BS value. Next, let's compare this to an ASTRAL tree.

ASTRID will allow us to estimate the species tree using differing methods: neighbour-joining (NJ), or distance based (fastme). You can designate it using the –m command, read about the alternative settings in the 'README.md' file in ASTRID, or on their website.

Get ASTRAL up and running and get your ducks in a row:

1. Open a new bash terminal, navigate to your ASTRAL folder, the general set-up is very similar to ASTRID, but you can check for help here: https://github.com/smirarab/ASTRAL/blob/master/astral-tutorial.md
2. ASTRAL seems a little more sensitive to the location of your files, so to make things easy, I make a folder in the Astral subdirectory within the ASTRAL directory, and put in a few things:
   - the gene trees: RAxML_bestTree.XXXX_ALL.txt
   - the bootstrap location file: bs-files
   - an additional folder with all the bootstrap files in it: they're labelled "RAxML_bootstrap.XXXX_L1"… L1 refers to locus one, and so on
3. Crack open the bs-files file in TextWrangler, we need to adjust the location of the files, so ASTRAL knows where to look for the BS trees.
   - drop one of the boostrap files into your terminal to get the name of the path, it should look something like: /Users/Ian/Google.Drive/R.Analyses/ASTRAL/Astral/Eulamprus/Gene.tree.BS/RAxML_bootstrap.T203_I
   - obviously the directory names will be different, but you get the idea
   - now change the file paths in the bs-files document to match the paths to the actual BS trees (by locus; use the option key and cmd+v) Cool, now we can run a quick and dirty species tree from the individual gene trees:

```
$ java -jar astral.4.10.8.jar -i [drag/drop input file: RAxML_bestTree_ALL.txt"]
```

As with ASTRID, you can define your output with

```
$ -o [name the output.tre]
```

Quick'n'dirty is nice, but let's take into account the BS replicates from our RAxML gene trees:

```
$ java -jar astral.4.10.8.jar -i [drag/drop input] -b [drag/drop "bs-file"] -o output.tre
```

This might take a little while. We're running 100 boostrap replicates on the boostrapped gene trees, so things are starting to get ugly.
* did you get an error message?
Something along the lines of "OutOfMemoryError: java heap space"? this means we don't have enough memory allocated to this analysis, so we've gotta define how much memory to let the terminal borrow. The more memory you can lend, the faster the analysis should go (theoretically), but you can't overlend, or risk crashing it all. So, check your activity monitor to see how much you can comfortably give, I used 6gb, but could probably give 12. Alternatively in a new terminal window, type "top" and check the "PhysMem" section to see what you can give (used vs. unused).

```
$ java -Xmx6000M -jar astral.4.10.8.jar -i [drag/drop input file] -b [drag/drop "bs-file"] -o output.tr
```

Here, the –Xmx6000M denotes 6000 mb of memory, change as you see fit.
That should sort you out, but his might take a little while. Running the 58 taxa Eulamprus dataset (378 loci) took about 80 min. when it's done, open in figtree. This file will include 102 trees. 1 tree per boostrap, 1 greedy algorithm tree, and the final consensus tree. I suggest copy/paste the last tree from this file into a new TextWrangler document, that's our go-to ASTRAL BS species tree.

*The current advantage of ASTRAL over ASTRID is that we're summarizing our genetic distance as branch lengths in coalescent units in ASTRAL. This can allow us to scale them to discrete units of time, to make a chronogram, though terminal branch lengths are not accurately estimated. ASTRID is certainly much faster, but NJ methods may come at the expense of phylogenetic accuracy.

Comparing Our Trees

For an exercise in why we're using coalescent methods, let's compare (visually) the ASTRAL species tree to the RAxML concatenated tree:

```r
install.packages('phytools')
library(phytools)

#you can read in your trees either as nexus or newick strings,
#just make sure that you do the same for both, or it jacks up the script

#read in your ASTRAL phylogeny:
astral<-read.nexus("astral.consensus.tre")

#read in your concatenated RAxML phylogeny
#if 'read.nex' doesn't work, try out 'read.tree':
raxml<-read.nexus("raxml.concatenated.tre")

#create an object ('co') of the trees facing one another
#with branches rotated to reduce crossed lines
co<-cophylo(astral, raxml, rotate=TRUE)

#plot the object, control font size and tipballs ('pts'), among other things here
plot(co, fsize=0.3, pts=FALSE)
```

boom, that was easy. export it so you can fiddle in illustrator or whathaveyou.

```r
#if you want to just look at the two trees plotted separately:
par(mfrow=c(1,2)) #set up a plot with two figures
#go back to 1 figure at a time with par(mfrow=c(1,1))
plot(astral) #plot first tree
plot(raxml) #plot second tree
```

We can use this same method to compare the topologies between coalescent methods too (ASTRAL vs. ASTRID), though we hope they're getting the same result.

As mentioned above, we're hoping that across all our methods, the topology does not change much. We expect that with this much data, the results should be the same[7]. However, extremely short internode distances as a result of rapid speciation can complicate accurate reconstruction. There are other reasons for possible conflict, but you can do your own reading. . .

[7]Tonini J, Moore A, Stern D, Shcheglovitova M & G, O. Concatenation and species tree methods exhibit statistically indistinguishable accuracy under a range of simulated conditions. PLOS Currents Tree of Life 1, doi:10.1371/currents.tol.34260cc27551a527b124ec5f6334b6be (2015).

The Coalescent in a Bayesian Framework

Alright, we've got ~400 loci, and we want to run our multispecies coalescent in *BEAST in BEAST2[8]. But slow down, at the moment, this is way too many markers to use in a true Bayesian framework under coalescent methods. Our problem won't be running the analysis, but likely reaching convergence, so let's sort through em and choose a subset:
Huw recommends a max of ~2000–3000 loci for BEAST2, to finish in a reasonable amount of time, I'll keep playing with numbers, but let's use this as a start. That means in the Eulamprus data set (29 samples; 58 with phased data) we max out around 50 loci. If your data set is bigger like the elapids (n=90; 180 phased) adjust accordingly (max ~20).

1. We'd like an estimate of informativeness on a per-locus basis, so we'll use PhyDesign[9] (full disclosure: this step is not essential, and will take some time, so just a heads-up)
   - found here: http://phydesign.townsend.yale.edu/
2. First step: turning our phylip alignments into Nexus files.
3. Grab your concatenated alignment of all loci, mine is in my "RAxML_2alleles" folder, and is called "T203_ConcatLoci.phylip" Drop this into a folder in Geneious.
4. Then "File->Export->Selected Documents->Nexus. Untick the box labelled "export as interleaved."
5. We've got our alignment, but it's not partitioned by locus, so open the file in TextWrangler. Then open the partition file titled something like "RAxML_ConcatLoci.charSets"
6. Copy the partitions, and paste them into the bottom of the alignment file. To keep this as brief as possible, just copy the formatting of the "PhyDesign.Nexus.nex" file that I've included in this WorkFlow folder.
7. The second input PhyDesign requires is an ultrametric tree. We haven't run our *BEAST analysis yet, nor have we translated the coalescent units of our ASTRAL tree, so we'll just fudge one with a constant relative rate using the R package diveRsity[10] and the "chronos" function:

```r
install.packages('diveRsity')
install.packages('ape')
library(diveRsity)
library(ape)

raxml<-read.tree("raxml.concatenated.tre")
ultra<-chronos(raxml, lambda=1)
plot(ultra)
write.tree(ultra, file="Eulamprus.ultra.info.tre")

citation('diveRsity')
```

8. We're almost there. Upload your alignment and your tree and hold your breath until PhyDesign is done running through your estimates (this will likely take a few hours).
9. Ok, PhyDesign is showing us the graphs of all of our loci, take note of any particularly funky ones (disproportionately high peaks near the present). Export the data from the option on the left side of the screen (>Profile data points).
10. In the datafile, create a new row for the average informativeness per locus. Average the column, sort the columns by these values (data->sort->Options->Sort left to right)

[8]Bouckaert, R. et al. BEAST 2: A Software Platform for Bayesian Evolutionary Analysis. PLoS Comput Biol 10, e1003537, doi:10.1371/journal.pcbi.1003537 (2014).

[9]López-Giráldez, F. & Townsend, J. P. PhyDesign: an online application for profiling phylogenetic informativeness. BMC Evol. Biol. 11, 1-4, doi:10.1186/1471-2148-11-152 (2011).

[10]Keenan, K., McGinnity, P., Cross, T. F., Crozier, W. W. & Prodöhl, P. A. diveRsity: An R package for the estimation and exploration of population genetics parameters and their associated errors. Methods in Ecology and Evolution 4, 782-788, doi:10.1111/2041-210X.12067 (2013).

Let’s get active in starBEAST

1. BEAST2 doesn’t like anything except nexus files, so for all the loci you want to use, you’ll have to change them from phylip format. I just drag/drop em into Geneious, and do a batch export as nexus.
2. Here’s the next methodological quandary: do we want (a) the most informative loci; (b) only fully sampled (no missing data) loci; or (c) the most informative, fully sampled loci?

3. The easiest way would be (b) chuck all the loci into Geneious, pick 50-100 that have complete sampling, and away you go. But for the sake of argument, I’ll show you how to do (a) and to a lesser extent (c).

```
## Intermission ##
```

Piggybacking on Moos: Using EAPhy[11]

1. Moos created a fantastic pipeline for QC’ing our sequences. We’re going to focus on two functions of this process (findSpeciesID & final_align) to improve our alignments for *BEAST.
2. *BEAST requires all taxa be included in all alignments, even if for a given locus that means a string of Ns the whole way. With EAPhy we’ll do just that—provide Ns for any samples that aren’t included in a given alignment, and we’ll also change the names of the samples to make them more palatable. (either fasta or nexus format is ok for Beauti)
3. I could bore you all day with specifics, but just follow Moos’s tutorial, and I’ve included examples of my input files in this folder as well.
4. EAPhy works in Python, and the easiest way to use this is via an interface. I use Canopy, which, like R Studio, makes using Python easy.
5. Once you’ve downloaded EAPhy and Canopy, mess around with Canopy a bit, to get comfortable. To avoid making this tutorial any longer, I minimize the general Canopy steps, so mess around and get familiar on your own.
6. Designate the ‘EAPhy-master’ folder as your working directory.
7. Copy and drop the folder full of fasta alignments into the ‘EAPhy-master’ folder (Batch Export your phylip files from Geneious as fasta)
8. In this fasta folder (example is “Eulamprus_fasta_alignments”), should be three additional files:
    a. ‘loci_list.txt’: a list of the names of all the alignment files
    b. ‘[insert group]_ID.txt’: a list of the sample ids found in the alignments
    c. ‘ID_changes.txt’: a tab-delimited file with the provided sample ids, and what you would like the names to be changed to.
9. We’re using phased (haplotype) data from the Lemmons. EAPhy digs haplotype data, but it must be provided with the suffix ‘h0’ and ‘h1’. Ours are labelled as ‘seq1’ and ‘seq2’, so:
    a. Open a terminal window, cd over to your fasta folder, and change all the alignments:

```
$ perl -pi -w -e ‘s/[old ending]/[new ending]/g;’ *.fasta

#This looks in all files ending with ‘.fasta’ for things ending with the ‘old ending’
#and replaces them with the ‘new ending’, here’s an example:
$ perl -pi -w -e ‘s/_seq1/_h0/g;’ *.fasta
```

10. Open up your EAPhy_config.py file in Canopy, and adjust the #Input files, see my example: ‘Eulamprus_config.py’

[11] Blom, M. P. K. EAPhy: A Flexible Tool for High-throughput Quality Filtering of Exon-alignments and Data Processing for Phylogenetic Methods. PLoS Currents 7, ecurrents.tol.75134257bd75134389c75134204bc75134251d75134226d75134242aa75139089f, doi:10.1371/currents.tol.75134257bd389c04bc1d26d42aa9089f (2015).

11. Because we're running through EAPhy for the first time, run a full analysis: 'complete = 1'
12. Things get tricky once we get to the # Parameter Settings. Our data has largely been covered by the Lemmons, so in theory we should just be able to plug'n'play our alignments. We will relax the alignment filtering to the point that it's pretty much asleep, this will pass the most loci through the filter, and allow us to use more for our *BEAST analysis. You can mess around with the filtering stringency on your own. Read up on the specifics.
    a. note, I set the 'max_insertion_cutoff' to 58, this avoids removing sections with insertions across a large number of taxa. I did this specifically for the Eulamprus data, because it's mostly intraspecific, you can make this more stringent, actually I recommend it.
    b. lastly, the 'number_missing_indivs_allowed'. I gave it a set of (0,6,12), this means it will return alignments with a max of 0, 6, 12 missing sequences, or 0, 3, 6 missing individuals, which means 20 individuals (highly unlikely). You can run different sets of loci to test effects of missing data, perhaps to look at branch-length/topology estimation.
13. Run this in Canopy by going to Tools->Canopy->Terminal. You can run the EAPhy pipeline outside of Canopy, in your own bash terminal, but make sure you're using a version of Python older than 3.0. I designated version 2.7 with the command:

```
alias python=/usr/local/bin/python2.7
#EAPhy gave me consistent errors on python v3.0, but Canopy automatically uses 2.7 anyway.
```

14. Run the analysis by navigating to your 'EAPhy-master' folder via 'cd' in the terminal. Then the command:

```
python [your configuration file.py]
#example:
python Eulamprus_config.py
```

15. Pull your final .nex alignments: results->[file]_out->haplotype->2.alignments->3.final_align

Back to starBEAST

Running starBEAST in BEAST2 is extremely similar to a standard BEAST run, but a great, must-read tutorial can be found here:
http://treethinkers.org/tutorials/divergence-time-estimation-using-beast/
I've also included it in the workflow folder as "Heath – 2014 – Divergence Time. . . "
1. Start by opening BEAUti, and navigating to File->Template->StarBeast
2. At the moment, my major holdup with *BEAST is actually BEAUti. It freezes up if I try and give it >200 loci, and I can't create the xml file. This could be partly assuaged by linking the site models, if PartitionFinder tells you it's ok. But generally, we should keep the trees and clock models unlinked. You can also get around this by building your xml file by hand.
3. Drop in your nexus alignments. And give it a number of generations is probably won't ever attain (say, 1 billion). This way, we can run it for a really long time, and sample the log files to ascertain when our parameters estimates have converged using RWeThereYet. At that point, we can quit the analysis, and happily look at our tree.

Fast Forward say, 100 Million generations (Diagnostics & Convergence)

So, we've let the analysis run for a couple days, and we want to have a look and see what's cooking:

To determine if our *BEAST analysis has run for enough generations and there has been sufficient mixing in our MCMC chain(s), we're going to have a look at our tree and log files in R We There Yet? (rwty)[12]
1. I hear you, you're saying "but why don't we just assess convergence in Tracer?" Well, Tracer is going to give you an estimate of the effective sample size (ESS—a measure of convergence) for the posterior/likelihood/prior, but it doesn't give you a measure of the topological convergence. RWTY does both, so let's just throw it all in there and we can get all the data necessary.

```r
install.packages('rwty')
library(rwty)
```

read in your *beast trees:

```r
#read in your *beast tre
sptrees<-load.trees("species.trees", type="nexus", format="*beast", logfile="T203_L6.log")
```

get a quick estimate of topological ESS values, aim >200:

```r
approx.ess<-topological.approx.ess(sptrees, burnin=2000)
```

analyze the trees and designate burnin:

```r
allplots<-analyze.rwty(sptrees, burnin=2000, window.size=100, treedist='RF')
```

the 'analyze.rwty' function above will tell you all the plots it's creating, but we've put em into the 'allplots' object, because we don't want it to print every single one of them (maybe you do, but I don't)
So, instead let's just have a look through the plots that might actually mean something. Most will have two plots (a trace and a distribution) for all parameters, make sure ESS>200 to determine sufficient MCMC mixing:

```r
#plot the estimate of the posterior:
allplots$posterior.trace

#plot the estimate of the likelihood:
allplots$likelihood.trace
```

[12]Warren, D., Geneva, A. & Lanfear, R. rwty: R We There Yet? Visualizing MCMC Convergence in Phylogenetics, https://cran.r-project.org/package=rwty (2016).

```
#plot the estimate of the prior:
allplots$prior.trace

#check the estimate of gene coalescence with "species"
allplots$speciescoalescent.trace

#check the estimate of topological convergence
allplots$topology.trace.plot #

#you've worked hard, have a pretty graph of the parameter estimates through your chain:
allplots$Chain.1.correlations #

#get a handle on the Birth/Death ratios
allplots$BirthDeath.t.Species.trace

#look at the Birth (speciation) rate estimate
allplots$birthRate2.t.Species.trace

#try to get an idea of Death (extinction) rate
allplots$relativeDeathRate2.t.Species.trace

#check the space the tree search has covered and converged upon:
allplots$treespace.heatmap
```

Explanations of these plots can be found in Dan Warren's tutorial.
You can find this information here: https://github.com/danlwarren/RWTY
2. If your analysis has finally converged, hooray.
3. Open up TreeAnnotator, chuck in your species.trees file, designate the amount of burnin (determined from RWTY), and you've got your *BEAST tree in hand. Compare it to your other coalescent and concatenated methods to see what kind of support you have across varied methods.

Using ALL of our Data

General feelings in phylogenetics are that more data = greater phylogenetic accuracy. Obviously there is a lot to take into consideration here, such as the quality of the data, and how we apply it, but in this molecular phylogenomics example, we want to use as much of our data as possible. We've already paid for it, so we might as well make it work for us.

Previously, we used subsets of our data in a multispecies-coalescent (MSC) approach. This is good, because the MSC is a more biologically accurate model for phylogenetic reconstruction. MSC methods are, however, time and computationally intesnsive, meaning we can't use ALL the data.

In contrast, we did manage to use ALL of our data in concatenation methods like RAxML. The shortcoming here is that concatenation smothers signal from different loci by forcing them into the same topology. Not biologically realistic, and certainly not ideal. But it is fast!

To get around these problems, we'll again use ASTRAL, our shortcut coalsecent analysis (SCA) of choice. ASTRAL is very sensitive to the quality of the input gene trees [13], so instead of using RAxML reconstructed gene trees, we'll use the *BEAST gene trees we made in the previous MSC step.

1. Start by running ALL of your genes through *BEAST in manageable sized chunks. This is largely dependent upon the number of "species" in your dataset, and the number of samples. I'll do 10-25 genes together at a time.
2. Once they've all converged, we'll start the process of assembling them all for use in ASTRAL.

*BEAST to ASTRAL Pipeline

1. Read in all the gene trees you want to handle in this step. You'll do this by setting your working directory to the folder you've stored them all in, and then identifying the pattern to find them:

```r
library(ape)
library(phytools)

files <- list.files(pattern="_species_final_align.trees", recursive=FALSE)
```

2. Loop through all the tree files, and use TreeAnnotator to make a consensus tree for each locus. Designate your burnin.

```r
for(i in 1:length(files)){
  input = files[i]
  burnin = 50
  #in BEAST2, this is a percentage, might need to adjust this script for BEAST1
  exe = "/Applications/BEAST_2.4.2/bin/treeannotator"
  #this is the location of treeannotator in your current BEAST directory

  filename <- tail(unlist(strsplit(input, "/")), 1)
  datapath <- gsub(filename, "", input)
  filename <- head(unlist(strsplit(input, "\\.")), 1)
  output <- paste(filename, "con", sep = ".")

  burn.in <- paste("-b",burnin)
  call <- paste(exe, burn.in, input, output)
  system(call)
  tr <- read.nexus(output)
```

[13] Mirarab, S. & Warnow, T. ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. Bioinformatics 31, doi:10.1093/bioinformatics/btv234 (2015).

```
  tr
}
```

3. Read in all the consensus files, so we can make a file that includes all of them.

```
con.files <- list.files(pattern=".con", recursive=FALSE)
```

4. Now loop through all the consensus gene tree files and write each one to a file called "all.consensus.genetrees" for use in ASTRAL

```
for(i in 1:length(con.files)){
  input = read.nexus(con.files[i])
  write.tree(input, file="all.consensus.genetrees", append=TRUE)
}
```

5. Then, you'll loop through all the original tree files again, and use logcombiner to remove the burnin from each locus.

```
for(i in 1:length(files)){
  input = files[i]
  burnin = 50
  #in BEAST2, this is a percentage, might need to adjust this script for BEAST1
  exe = "/Applications/BEAST_2.4.2/bin/logcombiner"
  #this is the location of logcombiner in your current BEAST directory

  filename <- tail(unlist(strsplit(input, "/")), 1)
  datapath <- gsub(filename, "", input)
  filename <- head(unlist(strsplit(input, "\\.")), 1)
  output <- paste(filename, "burned", sep = ".")

  burn.in <- paste("-b",burnin)
  call <- paste(exe, burn.in, "-log", input, "-o", output)
  system(call)
  tr <- read.nexus(output)
  tr
}
```

6. Read in these "post-burnin" tree files, so you can pull out a random sampling of them.

```
pb.files <- list.files(pattern=".burned", recursive=FALSE)
```

7. Loop through the post-burnin (pb) tree files and pull out 100 random trees, you'll use these as gene tree bootstraps for ASTRAL. Write them to a new file

```
for(i in 1:length(pb.files)){
  trees <- read.nexus(pb.files[i])
  treex <- sample(trees, size=100)
  write.tree(treex, paste(pb.files[i], "_random100.tre", sep=""))
}
```

8. Great, now you should have most of the files you'll need for your ASTRAL run.

Combining Data from More than One AHE Project

By scaling up the number of loci we use, we can resolve more difficult phylogenetic questions. This however, creates a problem when we want to combine data from different projects. Instead of being able to quickly drop a single species into the 4 gene alignments as you used to, we need tools to identify the same (orthologous) loci quickly from batches of dozens, hundreds or even thousands. To accomplish this, we'll use Reciprocal Blast Hits (RBH) which takes two sets of loci, blasts them against one another and back, then provides information regarding potentially congruent loci between the loci sets.

Start by following these instructions to install Galaxy:
1. Register an account at BOTH https://galaxyproject.org AND https://usegalaxy.org, and start following install directions found at https://galaxyproject.org/admin/get-galaxy/
2. Hopefully you've got 'git' installed. If so go to terminal and drop in:

```
$ git clone https://github.com/galaxyproject/galaxy.git
```

3. Galaxy is finicky about the version of Python used, so make sure you've got 2.7 available, even if 3.X is your default. If it comes up as '3.X' you'll need to make a work-around to direct it to version 2.7. instructions here: https://galaxyproject.org/admin/python/

```
$ python --version
```

4. Once you've got it up and running (properly directed), open a terminal window and direct it to Galaxy, then initiate it

```
$ cd /Users/YOUR.DIRECTORY.NAME/galaxy
# then execute galaxy
$ sh run.sh
# alternatively you can drag/drop 'run.sh' if it doesn't work for you
$ sh /Users/YOUR.DIRECTORY.NAME/galaxy/run.sh
# now close it down, because you have to activate admin access
$ ctrl+c
```

5. This is where I started to run into issues, so you might have to google some resolutions if you get into trouble. Make sure to follow the "Become an Admin" directions, including duplicating the 'config/galaxy.ini' file, and removing the '#' preceding the 'admin_users' line after you add your email address.
6. When you go to start Galaxy up again, you'll copy/paste http://127.0.0.1:8080/ into your browser, sign-in, and it should give you access to an 'Admin' tab at the top of the page. If you don't see it, close Galaxy server, and sign in again like this:

```
$ ctrl+c
$ sh run.sh --daemon
```

drop http://127.0.0.1:8080/ into your preferred browser.
sign in, and hopefully can view the Admin tab "'

Reciprocal BLAST

To find orthologs between our data sets, start by making alignments for each. Fasta Alignment A (FAA) will have each locus for loci set A, with the sample name as the locus name. Fasta Alignment B will be the same for loci set B. (examples in folder).

The first step is going to be putting together these FAA and FAB files. For our example, we have the taxon 'Acrochordus javanicus' as a sample in both loci sets. This is great, because it means we'll be looking for high-similarity matches when we do our reciprocal blast.

1. Direct over to the folder which holds all of the FAA loci alignments. Use grep to pull out all sequences which start with the unique sample name (here, Acrochordus) and paste them all into a new file.

```
$ cat *.txt |grep 'Acrochordus_javanicus' >> T222.Acrochordus.seqs.txt
```

2. This however, won't preserve the file names, which we need to identify the loci. So, pull the file names. This is complicated by the fact that Acrochordus may not be in every locus alignment, so specify our search to look for only the alignments that Acro is in.

```
$ find . -name "*T222*" -exec grep -l 'Acrochordus_javanicus' {} + > loci.names.txt
```

If you data is phased, you'll probably want to delete one haplotype before doing this, you can do this quickly in the terminal

```
$ awk 'NR % 2 == 0' file > newfile
# we've kept every second line
```

3. Now we need to be a little creative. Open the 'loci.names.txt' file, and paste the names in place of the sample names. Now with (option+dragdown) place a '>' before each sequence, and finally cmd+F and replace the tabs with a return. You can also do this with the 'Phylip to FASTA' R script that is in the next section.
4. Repeat this all for FAB.

Now we've made our two (FAA, FAB) files for reciprocal blasting in Galaxy with the Blast_RBH tool.
1. Upload the FAA and FAB files to your Galaxy instance.
2. Download the 'Blast_RBH' tool from the 'BLAST+' package to your toolshed in Galaxy.
3. After installing it in your toolshed, click the 'Analyze Data' tab, and look for the Blast_RBH (Recriprocal Blast) tool on the left.
4. Load the two alignments, and adjust settings as you see fit. I'm using the nucleotide sequences from the same organism, so I'll use 'megablast'.
5. Once the analysis has run, download your output file, and open it in excel, then save it as a .csv.
6. The following instructions are included in the "Muscle - Combining and Aligning.R" script included in this folder, but I'll walk through it here. Download and install 'Muscle', the command-line alignment tool.

```
source("https://bioconductor.org/biocLite.R")
biocLite("muscle")
library(muscle)
```

7. Put all of your alignments into a single folder, alongside your .csv file that explain which loci are orthologous. Then loop through, combining the orthologs into a single Phylip file for each locus.

```
overlap <- read.csv("loci.overlap.csv", header=T) # read in the congruence file
for (z in 1:nrow(overlap)) {
  locus.a <- overlap[z,1]
  locus.b <- overlap[z,2]

  file.a <- paste(locus.a, "_phylip.txt", sep="") # adjust to suit alignment file ending
  file.b <- paste(locus.b, ".phylip", sep="") # adjust to suit alignment file ending

  new.name <- paste("combined.",locus.a,"__",locus.b,".fasta", sep="")
  call <- paste("cat", file.a, file.b, ">", new.name)
  system(call)
}
```

8. Now the orthologues are combined, we need to transfer them from Phylip to Fasta format.

```
path = "/Users/Ian/Desktop/combined" # designate path to the COMBINED alignments
file.names <- dir(path, pattern =".phylip") # adjust to suit the ending
for (q in 4:length(file.names)) {
  # first remove the first line with the number of taxa and characters
  file <- paste(path, "/", file.names[q], sep="")
  short.file <- file.names[q]
  removed <- paste("sed -i '' '1d'", file)
  system(removed)

  #alternatively this can be done more gracefully with 'system2'
  #system2("sed", args=c("-i", "''", "'1d'", file))

  #now we'll add the 'fasta carrot'
  arrow <- paste("sed -i '' 's/^/>/'", file)
  system(arrow)

  #then insert a return for each tab and save the file with '.fasta' ending
  fasta.name <- paste(short.file, ".fasta", sep="")
  returned <- paste("tr '\t' '\n' <", file, ">", fasta.name)
  system(returned)
}
```

9. Finally, we'll send each 'combined' fasta file to Muscle to be aligned. Depending on how many loci you have, this could take a while.

```
fasta.files <- dir(path, pattern=".fasta") # desginate path to the COMBINED FASTA files
for(i in 1:length(fasta.files)) {
  locus <- paste(path,"/",fasta.files[1], sep="")
  new.name <- paste("aligned.", fasta.files[1], sep="")
  call <- paste("/Users/Ian/Desktop/combined/muscle3.8.31_i86darwin64",
                "-in", locus, "-out", new.name)
  system(call)
}
```

10. All of the matching loci should now be aligned with each other appropriately. Now you can analyze this combined dataset.

Special Considerations

In the Australian elapid data, and to a lesser extent in the python data, we have samples which did not amplify for the vast majority of loci (<15% coverage). We need to remove these sequences prior to our analyses, or at minimum remove them from an additional run, so we can compare results between a phylogeny with this missing data and without it. We'll do this in the terminal:

```
#Navigate to the folder with our alignments,
$ sed -i ‘’ -e ‘/[taxon name]/d’ *.txt
```

This deletes the entire line with the offending sample, from all files in the directory ending in .txt