Phylogenetic Inference using RevBayes

Total-evidence Dating under the FBD Model

Tracy A. Heath, April Wright, and Walker Pett

1 Introduction

Ronquist et al. (2012)

Exercise is in Section 3.

1.1 Models

1.1.1 Lineage Diversification and Sampling

Birth-death processes and FBD

1.1.2 Sequence Evolution

In this tutorial, our molecular data are best analyzed with a General Time-Reversible model of sequence evolution and gamma-distributed rate heterogeneity. This model of sequence evolution is covered thoroughly in the RB_CTMC_Tutorial.

1.1.3 Morphological Character Change

The incorporation of morphological data from fossils into phylogenetic analyses allows us to directly observe organisms that existed in the past. In 2001, Paul Lewis described a model, the Mk model, to allow for the incorporation of morphology into likelihood and Bayesian analyses. The Mk model uses a generalized Jukes-Cantor matrix. This introduces a number of assumptions. One key assumption is that characters exhibit symmetrical change - that a given character is as likely to transition from a one state to another as it is to reverse. For example, this assumption applied to a binary character would mean that a change from a 0 state to a 1 state is as likely as a change from a 1 state to a 0 state.

In the tutorial, we will relax this assumption through the use of a prior on stationary state frequencies. This prior is visualized in Fig. 1. All of the morphological data we are working with in this tutorial are binary. Therefore, we can draw the stationary state frequencies of our 0 and 1 characters from a Beta distribution. Using the stationary state frequencies drawn from the Beta distribution, we can then create new Q-matrices that allow for asymmetrical change in characters states (i.e., for changing from state 0 to state 1 to be more likely than changing from state 1 to state 0, or vice versa). We can do multiple draws from the Beta distribution, creating multiple different Q-matrices per datset. This allows for heterogeneity in character change asymmetry across the dataset. Likewise, we will use the Gamma distributed rate heterogeneity described in (other tutorial) to allow for different characters to evolve at different rates.

Because of the way morphological data are collected, we need to exercise caution in how we model the data. Traditionally, phylogenetic trees were built from morphological data using parsimony. Therefore, characters

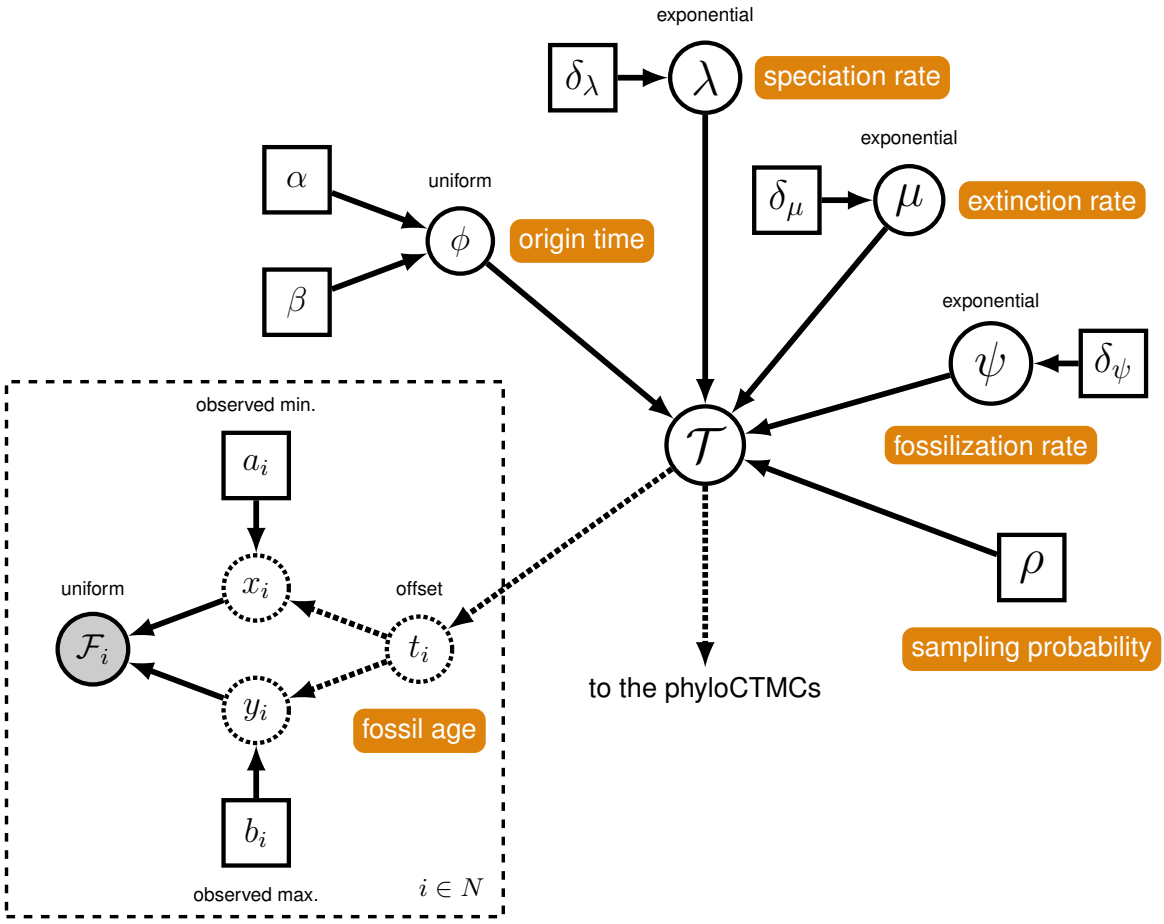

Figure 1: A graphical model of the fossilized birth death process. In this model, the speciation, extinction and fossilization rates are drawn from exponential distributions, while the origin time and fossil ages are uniformly distributed.

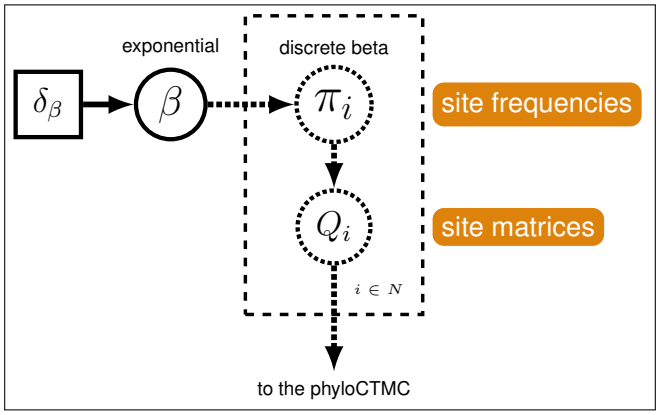

Figure 2: A graphical model of the Beta-distributed frequencies across sites model. In this model, stationary state frequencies are drawn from a discrete Beta distribution and used to generate Q-matrices that allow character change asymmetry.

were collected to be parsimony informative - that is, to be useful for discriminating between phylogenetic hypotheses under the maximum parsimony criterion. This means that many morphological datasets do not

contain invariant characters or autapomorphies, as these cannot be used to discriminate among hypotheses under parsimony. By excluding these low rate of evolution characters, the estimated branch lengths can be inflated. Therefore, it is important to use models that can condition on the coding bias. RevBayes has two ways of doing this: one is used for datasets in which only parsimony informative characters are observed; the other is for datasets in which parsimony informative characters and non-parsimony informative variable characters (such as autapomorphies) are observed.

1.1.4 Lineage-Specific Substitution Rates

Clocks (Zuckerkandl and Pauling 1962) and relaxing them

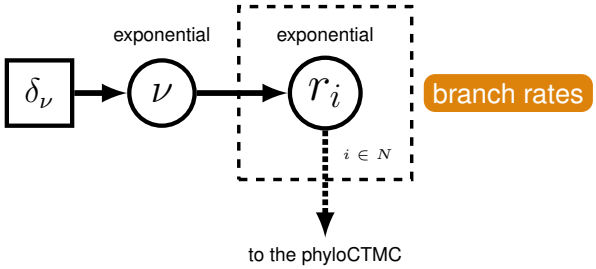

Figure 3: A graphical model of the uncorrelated exponential relaxed clock model. In this model, the clock rate on each branch is identically and independently distributed according to an exponential distribution with mean drawn from an exponential hyperprior distribution.

2 Prerequisites

What do you need to know before doing this?

2.1 Requirements

We assume that you have read and hopefully completed the following tutorials:

- RB_Getting_Started
- RB_Basics_Tutorial

Note that the RB_Basics_Tutorial introduces the basic syntax of `Rev` but does not cover any phylogenetic models. You may skip the RB_Basics_Tutorial if you have some familiarity with `R`. We tried to keep this tutorial very basic and introduce all the language concepts on the way. You may only need the RB_Basics_Tutorial for a more in-depth discussion of concepts in `Rev`.

3 Exercise: Estimating the Phylogeny and Divergence Times of Fossil and Extant Bears

Information about the exercise, citations for the data, questions.

3.1 Data files

We provide the data files which we will use in this tutorial. You may want to use your own data instead. In the **`data`** folder, you will find the following files

- **bears_taxa.tsv**: a tab-separated table listing every bear species (both fossil and extant) and their occurrence dates. For extant taxa, the occurrence date is `0.0` (*i.e.,* the present) and for fossil species, the occurrence date is equal to the mean of the age range (the ranges are defined in a separate file).
- **bears_cytb.nex**: an alignment in NEXUS format of 1,000 bp of cytochrome-b sequences for 10 bear species. This alignment includes 8 living bears and 2 extinct sub-fossil bears.
- **bears_morphology.nex**: a matrix of 62 discrete, binary (coded `0` or `1`) morphological characters for 18 species of fossil and extant bears.
- **bears_fossil_intervals.tsv**: a tab-separated table containing the age ranges (minimum and maximum in millions of years) for 14 fossil bears.

3.2 Getting Started

On your own computer, create a directory called `RB_TotalEvidenceDating_FBD_Tutorial` (or any name you like).

In this directory download and unzip the archive containing the data files: `data.zip`.

Additionally, create a new directory (in **RB_TotalEvidenceDating_FBD_Tutorial**) called **scripts**

When you execute `RevBayes` in this exercise, you will do so within the main directory you created (**RB_TotalEvidenceDating_FBD_Tutorial**).

3.3 Creating `Rev` Files

For complex models and analyses, it is best to create `Rev` script files that will contain all of the model parameters, moves, and functions. In this exercise, you will work primarily in your text editor and create a set of modular files that will be easily managed and interchanged. You will write the following files from scratch in a text editor[1] and save them in the **scripts** directory:

- **mcmc_TEFBD.Rev**: the master `Rev` file that loads the data, the separate model files, and specifies the monitors and MCMC sampler.
- **model_FBDP_TEFBD.Rev**: specifies the model parameters and moves required for the fossilized birth-death prior on the tree topology, divergence times, fossil occurrence times, and diversification dynamics.
- **model_UCExp_TEFBD.Rev**: specifies the components of the uncorrelated exponential model of lineage-specific substitution rate variation.
- **model_GTRG_TEFBD.Rev**: specifies the parameters and moves for the general-time reversible model of sequence evolution with gamma-distributed rates across sites (GTR+Γ).
- **model_Morph_TEFBD.Rev**: specifies the model describing discrete morphological character change (binary characters) under a strict morphological clock.

All of the files that you will create are also provided in the `RevBayes` tutorial repository[2]. Please refer to these files to verify or troubleshoot your own scripts.

[1] If you do not already have a good text editor that you like, we recommend one that has features for syntax coloring, easy navigation between different files, line numbers, etc. A good option is Sublime Text, which is available for Mac OSX, Windows, and Linux.

[2] https://github.com/revbayes/revbayes_tutorial/tree/master/RB_TotalEvidenceDating_FBD_Tutorial/scripts

3.4 Start the Master Rev File and Import Data

Open your text editor and create the master Rev file called mcmc_TEFBD.Rev in the **scripts** directory.

Enter the Rev code provided in this section in the new model file.

The file you will begin in this section will be the one you load into RevBayes when you've completed all of the components of the analysis. In this section you will begin the file and write the Rev commands for loading in the taxon list and managing the data matrices. Then, starting in Section 3.5, you will move on to writing module files for each of the model components. Once the model files are complete, you will return to editing **mcmc_TEFBD.Rev** and complete the Rev script with the instructions given in Section 3.9.

3.4.1 Load Taxon List

Begin the Rev script by loading in the list of taxon names from the **bears_taxa.tsv** file using the **readTaxonData()** function.

```
taxa <- readTaxonData("data/bears_taxa.tsv")
```

This function reads a tab-delimited file and creates a variable called **taxa** that is a list of all of the taxon names relevant to this analysis. This list includes all of the fossil and extant bears.

3.4.2 Load Data Matrices

RevBayes uses the function **readDiscreteCharacterData()** to load a data matrix to the workspace from a formatted file. This function can be used for both molecular sequences and discrete morphological characters.

Load the cytochrome-b sequences from file and assign the data matrix to a variable called **cytb**.

```
cytb <- readDiscreteCharacterData("data/bears_cytb.nex")
```

Next, import the morphological character matrix and assign it to the variable **morpho**.

```
morpho <- readDiscreteCharacterData("data/bears_morphology.nex")
```

3.4.3 Add Missing Taxa

In the descriptions of the files in Section 3.1, we mentioned that the two data matrices have different numbers of taxa. Thus, we must add any taxa that are not found in the molecular (**cytb**) partition (*i.e.,* are only found in the fossil data) to that data matrix as missing data, and do the same with the

morphological data partition (**morpho**). In order for all the taxa to appear on the same tree, they all need to be part of the same dataset, as opposed to present in separate datasets. This ensures that there is a unified taxon set that contains all of our tips.

```
cytb.addMissingTaxa( taxa )
morpho.addMissingTaxa( taxa )
```

3.4.4 Create Helper Variables

Before we begin writing the Rev scripts for each of the model components, we need to instantiate a couple "helper variables" that will be used by downstream parts of our model specification files. These variables will be used in more than one of the module files so it's best to initialize them in the master file.

Create a new constant node called **n_taxa** that is equal to the number of species in our analysis (22).

```
n_taxa <- taxa.size()
```

Next, create a *workspace variable* called **mvi**. This variable is an iterator that will build a vector containing all of the MCMC moves used to propose new states for every stochastic node in the model graph. Each time a new move is added to the vector, **mvi** will be incremented by a value of **1**.

```
mvi = 1
```

One important distinction here is that **mvi** is part of the RevBayes workspace and not the hierarchical model. Thus, we use the workspace assignment operator **=** instead of the constant node assignment **<-**.

Save your current working version of **mcmc_TEFBD.Rev** in the **scripts** directory.

We will now move on to the next Rev file and will complete **mcmc_TEFBD.Rev** in Section 3.9.

3.5 The Fossilized Birth-Death Process

Open your text editor and create the fossilized birth-death model file called **model_FBDP_TEFBD.Rev** in the **scripts** directory.

Enter the Rev code provided in this section in the new model file.

3.5.1 Speciation and Extinction Rates

Two key parameters of the FBD process are the speciation rate (the rate at which lineages are added to the tree) and the extinction rate (the rate at which lineages are removed from the tree). We'll place

exponential priors on both of these values. Each parameter is assumed to be drawn independently from a different exponential distribution with rates δ_λ and δ_μ respectively (see Fig. 1). Here, we will assume that $\delta_\lambda = \delta_\mu = 10$. Note that an exponential distribution with $\delta = 10$ has an expected value (mean) of 1/10.

```
speciation_rate ~ dnExponential(10)
extinction_rate ~ dnExponential(10)
```

For every stochastic node we declare, we must also specify proposal algorithms (called *moves*) to sample the value of the parameter in proportion to its posterior probability. If a move is not specified for a stochastic node, then it will not be estimated, but fixed to its initial value.

The rate parameters for extinction and speciation are both real, positive numbers (*i.e.,* floating point variables). For both of these nodes, we will use a scaling move (**mvScale()**), which proposes multiplicative changes to a parameter. Many moves also require us to set a *tuning value*, which is called **lambda** for **mvScale()**, which determine the size of the proposed change. Here, we will use three scale moves for each parameter with different values of lambda. By using multiple moves for a single parameter, we will improve the mixing of the Markov chain.

```
moves[mvi++] = mvScale(speciation_rate, lambda=0.01, weight=1)
moves[mvi++] = mvScale(speciation_rate, lambda=0.1, weight=1)
moves[mvi++] = mvScale(speciation_rate, lambda=1.0, weight=1)

moves[mvi++] = mvScale(extinction_rate, lambda=0.01, weight=1)
moves[mvi++] = mvScale(extinction_rate, lambda=0.1, weight=1)
moves[mvi++] = mvScale(extinction_rate, lambda=1, weight=1)
```

In addition to the speciation (λ) and extinction (μ) rates, we may also be interested in inferring diversification ($\lambda - \mu$) and turnover (μ/λ). Since these parameters can be expressed as a deterministic transformation of the speciation and extinction rates, we can monitor their values by creating two deterministic nodes using the **:=** operator.

```
diversification := speciation_rate - extinction_rate
turnover := extinction_rate/speciation_rate
```

3.5.2 Probability of Sampling Extant Taxa

All extant bears are represented in this dataset. Therefore, we will fix the probability of sampling an extant lineage (ρ) to 1.

```
rho <- 1.0
```

Because ρ is a constant node (assigned using the **<-** operator), we do not have to assign a move to this parameter.

3.5.3 The Fossil Sampling Rate

Since our data set includes serially sampled lineages, we must also account for the rate of sampling back in time. This is the fossil sampling (or recovery) rate (ψ), which we will instantiate as a stochastic node. As with the speciation and extinction rates (Sect. 3.5.1), we will use an exponential prior on this parameter and use scale moves to sample values from the posterior distribution.

```
psi ~ dnExponential(10)
moves[mvi++] = mvScale(psi, lambda=0.01, weight=1)
moves[mvi++] = mvScale(psi, lambda=0.1, weight=1)
moves[mvi++] = mvScale(psi, lambda=1, weight=1)
```

3.5.4 The Origin Time

Under the FBD model, the process is can be conditioned on the age of the origin, or the age of the root. In our case, we will condition on the origin time of bears, and we will specify a uniform distribution on the origin age. For this parameter, we will use a sliding window move. A sliding window samples a parameter uniformly within an interval (defined by the half-width **delta**). Sliding window moves can be tricky for small values, as the window may overlap zero. However, for parameters such as the origin age, there is little risk of this being an issue.

```
origin_time ~ dnUnif(37.0, 55.0)
moves[mvi++] = mvSlide(origin_time, delta=0.01, weight=5.0)
moves[mvi++] = mvSlide(origin_time, delta=0.1, weight=5.0)
moves[mvi++] = mvSlide(origin_time, delta=1, weight=5.0)
```

3.5.5 The FBD Distribution

All the parameters of the FBD process have now been specified. The next step is to use these parameters to define the FBD tree prior distribution.

```
fbd_dist = dnFBDP(origin=origin_time, lambda=speciation_rate, mu=extinction_rate, psi=
   psi, rho=rho, taxa=taxa)
```

3.5.6 Clade Constraints

Note that we created the distribution as a workspace variable using the workspace assignment operator **=**. This is because we still need to include a topology constraint in our final specification of the tree prior. Specifically, we do not have any morphological or molecular data for the fossil species *Ursus abstrusus*. Therefore, in order to use the age of this fossil as a calibration, we need to specify to which clade it belongs. In this case, *Ursus abstrusus* belongs to the subfamily Ursinae, so we define a clade for Ursinae including *Ursus abstrusus*.

```
clade_ursinae = clade("Melursus_ursinus", "Ursus_arctos", "Ursus_maritimus",
                      "Helarctos_malayanus", "Ursus_americanus", "Ursus_thibetanus",
                      "Ursus_abstrusus", "Ursus_spelaeus")
```

Then we can specify the final constrained tree prior distribution by creating a vector of constraints, and providing it along with the workspace FBD distribution to the constrained topology distribution. Here we use the stochastic assignment operator ~ to create a stochastic node for our tree variable.

```
constraints = v(clade_ursinae)
fbd_tree ~ dnConstrainedTopology(fbd_dist, constraints=constraints)
```

3.5.7 Moves on the Tree Topology and Node Ages

Next, in order to sample from the posterior distribution of trees, we need to specify moves that propose changes to the topology and node times. We also include a proposal that toggles whether a fossil is a "sampled ancestor" - a fossil species that has sampled descendants present on the tree. When conditioning on the origin time, we also need to explicitly sample the root age.

```
moves[mvi++] = mvFNPR(fbd_tree, weight=15.0)
moves[mvi++] = mvCollapseExpandFossilBranch(fbd_tree, origin_time, weight=6.0)
moves[mvi++] = mvNodeTimeSlideUniform(fbd_tree, weight=40.0)
moves[mvi++] = mvRootTimeSlideUniform(fbd_tree, origin_time, weight=5.0)
```

3.5.8 Sampling Fossil Occurrence Ages

Next we need to account for uncertainty in the age estimates of our fossils. We do this by assuming each fossil can occur with uniform probability anywhere within its observed stratigraphic range. Stratigraphic range data for each fossil species is provided in the file **bears_fossil_intervals.tsv**. First, we read this file into a matrix, then we loop over this matrix and for each fossil create a uniform random variable.

This is somewhat different from the typical approach to node calibration. Here, instead of treating the calibration density as an additional prior distribution on the tree, we treat it as the *likelihood* of our fossil data given the tree parameter. Specifically, we assume the likelihood of a particular fossil observation is equal to one if the fossil's inferred occurrence time falls within its observed stratigraphic range, and zero otherwise. This likelihood is proportional any uniform density that is non-zero when the occurrence time falls within the observed interval. For example, we'll use a random variable uniformly distributed in the interval offset by the fossil occurrence time, and we'll include this variable in the likelihood by clamping it at zero.

```
fossil_intervals = readDataDelimitedFile(file="data/bears_fossil_intervals.tsv",
    header=true)
```

```
for(i in 1:fossil_intervals.size())
{
    taxon = fossil_intervals[i][1]
    obs_min = fossil_intervals[i][2]
    obs_max = fossil_intervals[i][3]

    t[i] := tmrca(fbd_tree, clade(taxon))

    fossil[i] ~ dnUniform(t[i] - obs_max, t[i] - obs_min)
    fossil[i].clamp( 0 )
}
```

```
moves[mvi++] = mvFossilTimeSlideUniform(fbd_tree, origin_time, weight=5.0)
```

3.5.9 Monitoring Parameters of Interest using Deterministic Nodes

Finally, we can also create deterministic nodes for other quantities we might be interested in monitoring. We will define a deterministic node for the number of fossils in the tree that are sampled ancestors. We will also define a node for the age of the crown group of extant bears. In addition, we can monitor the sampled age of a particular fossil.

```
num_samp_anc := fbd_tree.numSampledAncestors()

clade_extant = clade("Ailuropoda_melanoleuca","Tremarctos_ornatus","Melursus_ursinus",
                  "Ursus_arctos","Ursus_maritimus","Helarctos_malayanus",
                  "Ursus_americanus","Ursus_thibetanus")
age_extant := tmrca(fbd_tree, clade_extant)

age_Kretzoiarctos_beatrix := tmrca(fbd_tree, clade("Kretzoiarctos_beatrix"))
```

3.6 The Uncorrelated Exponential Relaxed-Clock Model

Open your text editor and create the lineage-specific branch-rate model file called `model_UCExp_TEFBD.Rev` in the **`scripts`** directory.

Enter the `Rev` code provided in this section in the new model file.

For our hierarchical, uncorrelated exponential relaxed clock model, we first define the mean branch rate as an exponential random variable. Then, we specify some scale proposal moves on the mean rate parameter.

```
branch_rates_mean ~ dnExponential(10.0)

moves[mvi++] = mvScale(branch_rates_mean, lambda=0.01, weight=1.0)
moves[mvi++] = mvScale(branch_rates_mean, lambda=0.1, weight=1.0)
moves[mvi++] = mvScale(branch_rates_mean, lambda=1.0, weight=1.0)
```

Before creating a rate parameter for each branch, we need to get the number of branches in the tree. For rooted trees with n taxa, the number of branches is $2n - 2$.

```
n_branches <- 2 * n_taxa - 2
```

Then, use a for loop to define a rate for each branch. The branch rates are independently and identically exponentially distributed with mean equal to the mean branch rate parameter we specified above. For each rate parameter we also create scale proposal moves.

```
for(i in 1:n_branches){
    branch_rates[i] ~ dnExp(1/branch_rates_mean)
    moves[mvi++] = mvScale(branch_rates[i], lambda=1.0, weight=1.0)
    moves[mvi++] = mvScale(branch_rates[i], lambda=0.1, weight=1.0)
    moves[mvi++] = mvScale(branch_rates[i], lambda=0.01, weight=1.0)
}
```

Lastly, we use a vector scale move to propose changes to all branch rates simultaneously. This way we can sample the total branch rate independently of each individual rate, which can improve mixing.

```
moves[mvi++] = mvVectorScale(branch_rates, lambda=0.01, weight=4.0)
moves[mvi++] = mvVectorScale(branch_rates, lambda=0.1, weight=4.0)
moves[mvi++] = mvVectorScale(branch_rates, lambda=1.0, weight=4.0)
```

3.7 The General Time Reversible + Gamma Model of Nucleotide Sequence Evolution

Open your text editor and create the molecular substitution model file called `model_GTRG_TEFBD.Rev` in the **`scripts`** directory.

Enter the `Rev` code provided in this section in the new model file.

For our nucleotide sequence evolution model, we need to define a general time reversible (GTR) substitution matrix. A nucleotide GTR matrix is defined by a set of 4 stationary frequencies, and 6 exchangeability rates. We create stochastic nodes for these variables, each drawn from a uniform Dirichlet prior distribution.

```
sf_hp <- v(1,1,1,1)
sf ~ dnDirichlet(sf_hp)

er_hp <- v(1,1,1,1,1,1)
er ~ dnDirichlet(er_hp)
```

We need special moves to propose changes to a Dirichlet random variable, also known as a simplex (a vector constrained sum to one). Here, we use a "Simplex Element Scale" move, which scales a single element of a simplex and then renormalizes the vector to sum to one. The tuning parameter **alpha** specifies how conservative the proposal should be, with larger values of **alpha** leading to proposals closer to the current value.

```
moves[mvi++] = mvSimplexElementScale(er, alpha=10.0, weight=5.0)
moves[mvi++] = mvSimplexElementScale(sf, alpha=10.0, weight=5.0)
```

Then we can define a deterministic node for our GTR substitution matrix using the special GTR matrix function.

```
Q_cytb := fnGTR(er,sf)
```

Next, in order to model gamma-distributed rates across, we create an exponential parameter α for the shape of the gamma distribution, along with scale proposals.

```
alpha_cytb ~ dnExponential( 1.0 )

moves[mvi++] = mvScale(alpha_cytb, lambda=0.01, weight=1.0)
moves[mvi++] = mvScale(alpha_cytb, lambda=0.1, weight=1.0)
moves[mvi++] = mvScale(alpha_cytb, lambda=1, weight=1.0)
```

Then we create a Gamma(α, α) distribution, discretized into 4 rate categories using the "Discretize Gamma" function. Here, **rates_cytb** is a deterministic vector of rates computed as the mean of each category.

```
rates_cytb := fnDiscretizeGamma( alpha_cytb, alpha_cytb, 4 )
```

Finally, we can create the phylogenetic continuous time markov chain (PhyloCTMC) distribution for our sequence data, including the gamma-distributed site rate categories, as well as the branch rates defined as part of our exponential relaxed clock. We set the value of this distribution equal to our observed data and identify it as a static part of the likelihood using the **clamp** method.

```
phySeq ~ dnPhyloCTMC(tree=fbd_tree, Q=Q_cytb, siteRates=rates_cytb, branchRates=
    branch_rates, type="DNA")
phySeq.clamp(cytb)
```

3.8 Modeling the Evolution of Binary Morphological Characters

Open your text editor and create the morphological character model file called `model_Morph_TEFBD.Rev` in the **scripts** directory.

Enter the `Rev` code provided in this section in the new model file.

As stated in the introduction, we will use a prior on state frequencies to relax the key assumption of the Mk model. Because we are working with binary data, we can use a discrete Beta distribution to describe the variation in stationary state frequencies across characters. The Beta distribution has two parameters, α and β which describe its shape. For simplicity, we will assume that $\alpha = \beta$. The below code draws a value for β from an exponential distribution and places a move on it.

```
beta_hp ~ dnExponential( 1.0 )

moves[mvi++] = mvScale(beta_hp, lambda=1, weight=1.0 )
moves[mvi++] = mvScale(beta_hp, lambda=0.1, weight=1.0 )
moves[mvi++] = mvScale(beta_hp, lambda=0.01, weight=1.0 )
```

Next, we'll create a vector containing four different site stationary state frequencies. This is similar to allowing gamma-distributed rate variation across sites. We will then use these stationary frequencies to generate a set of new Q-matrices which do not enforce the assumptions of the same transition rates at every site and equal forward and backwards transition rates.

```
beta_cats := fnDiscretizeBeta( beta_hp, beta_hp, 4)
for(i in 1:beta_cats.size())
{
    Q_morpho[i] := fnFreeBinary(v(1-beta_cats[i], beta_cats[i]))
}
```

As in the molecular data partition, we will allow gamma-distributed rate heterogeneity among sites.

```
alpha_morpho ~ dnExponential( 1.0 )
rates_morpho := fnDiscretizeGamma( alpha_morpho, alpha_morpho, 4 )
```

```
moves[mvi++] = mvScale(alpha_morpho, lambda=0.01, weight=1.0)
moves[mvi++] = mvScale(alpha_morpho, lambda=0.1, weight=1.0)
moves[mvi++] = mvScale(alpha_morpho, lambda=1, weight=1.0)
```

Each data partition has to have a clock rate. For simplicity, we will assume a strict clock rate drawn from an exponential distribution.

```
clock_morpho ~ dnExponential(1.0)

moves[mvi++] = mvScale(clock_morpho, lambda=0.01, weight=4.0)
moves[mvi++] = mvScale(clock_morpho, lambda=0.1, weight=4.0)
moves[mvi++] = mvScale(clock_morpho, lambda=1, weight=4.0)
```

As in our molecular data partition, we now combine our data and our model. There are some unique aspects to doing this for morphology.

You will notice that we have an option called **coding**. This option allows us to condition on biases in the way the morphological data were collected (ascertainment bias). The option **coding=variable** specifies that we should correct for coding only variable characters (discussed in **??**).

We use the flag **siteMatrices=true** to indicate that we are providing multiple Q matrices generated as a function of our state frequency variation model.

```
phyMorpho ~ dnPhyloCTMC(tree=fbd_tree, siteRates=rates_morpho, branchRates=
    clock_morpho, Q=Q_morpho, type="Standard", coding="variable", siteMatrices=true)
phyMorpho.clamp(morpho)
```

3.9 Complete MCMC File

Return to the master `Rev` file you created in Section 3.4 called `mcmc_TEFBD.Rev` in the **scripts** directory.

Enter the `Rev` code provided in this section in this file.

3.9.1 Source Model Scripts

This step will load in the model scripts we have written in the text editor.

```
source("scripts/model_FBDP_TEFBD.Rev")

source("scripts/model_UCExp_TEFBD.Rev")

source("scripts/model_GTRG_TEFBD.Rev")

source("scripts/model_Morph_TEFBD.Rev")
```

3.9.2 Create Model Object

```
mymodel = model(sf)
```

3.9.3 Specify Monitors and Output Filenames

```
mni = 1
monitors[mni++] = mnModel(filename="output/bears.log", printgen=10)
monitors[mni++] = mnFile(filename="output/bears.trees", printgen=10, fbd_tree)
monitors[mni++] = mnScreen(printgen=10, age_extant, num_samp_anc, origin_time)
```

3.9.4 Set up the MCMC

```
mymcmc = mcmc(mymodel, monitors, moves)

mymcmc.run(generations=10000)
```

Save and close all files.

3.10 Run it

```
./rb
```

Execute the MCMC analysis:

```
source("scripts/mcmc_TEFBD.Rev")
```

```
   Processing file "scripts/mcmc_TEFBD.Rev"
   Successfully read one character matrix from file 'data/bears_cytb.nex'
   Successfully read one character matrix from file 'data/bears_morphology.nex'
   Processing file "scripts/model_FBDP_TEFBD.Rev"
   Processing of file "scripts/model_FBDP_TEFBD.Rev" completed
   Processing file "scripts/model_UCExp_TEFBD.Rev"
   Processing of file "scripts/model_UCExp_TEFBD.Rev" completed
   Processing file "scripts/model_GTRG_TEFBD.Rev"
   Processing of file "scripts/model_GTRG_TEFBD.Rev" completed
   Processing file "scripts/model_Morph_TEFBD.Rev"
   Processing of file "scripts/model_Morph_TEFBD.Rev" completed

   Running MCMC simulation
   This simulation runs 1 independent replicate.
   The simulator uses 163 different moves in a random move schedule with 267 moves per iteration

Iter      |     Posterior  |    Likelihood  |        Prior  |    age_extant  |  num_samp_anc  |   origin_time  |   elapsed  |       ETA  |
-------------------------------------------------------------------------------------------------------------------------------------------
0         |      -8174.01  |       -8053.8  |      -120.209  |       34.8641  |            0   |       44.4332  |  00:00:00  |  --:--:--  |
10        |      -4654.95  |       -4611.2  |      -43.7495  |       4.32618  |            7   |       45.4494  |  00:00:01  |  --:--:--  |
20        |      -4294.05  |      -4266.91  |      -27.1443  |       4.58804  |            7   |       46.5636  |  00:00:01  |  00:08:19  |
30        |      -4267.35  |      -4233.41  |       -33.94  |        6.8467  |            6   |       45.9177  |  00:00:02  |  00:11:04  |
40        |      -4226.63  |      -4188.32  |      -38.3037  |       6.40484  |            8   |       44.3696  |  00:00:02  |  00:08:18  |
...
```

3.11 Summarize Your Results

3.11.1 Evaluate MCMC

For this part of the tutorial, we will use a software called Tracer. Tracer allows researchers to visualize the MCMC sample for any parameters in the log file. Start Tracer by double-clicking on the Tracer executeable. Click the plus sign on the left-hand side of the screen to add your log file (see Fig. 6). Navigate to your log file, and click "OK".

Immediately upon loading, you will view the "Estimates" window, which shows the actual values sampled in the MCMC. In this window, the statistics on the left-hand side are the mean of the distribution and the Effective Sample Size (ESS) for that parameter. Parameters in a phylogenetic model may be correlated. The ESS is meant to provide a representation of the number of independent draws represented by the MCMC sample. This quantity will typically be much smaller than the number of generations of the chain. Poor values for the ESS will be colored red. You will likely see a lot of red values in the log file you made today because we did not run the chain very long. The data used to make the screenshots can be found in the bears_long.zip file in the output directory.

Take a look at some of the parameters. Are there any that have really low ESS? Why do you think that might be?

Next, we can click over to the Trace window. This window shows us the estimates for a given parameter, ordered by position in the chain. The left side of the chain often has a portion of the chain shaded out. This is referred to as 'burn-in'. The idea of burn-in is that an MCMC may not immediately begin sampling from the true distribution of values, particularly if the starting condition of the chain is poor. Therefore, these values are often discarded (or 'burned').

The trace window allows us to evaluate how well our chain is sampling parameter space. The output in Figure 4 is a fairly good trace - there is no consistent pattern or trend in the samples, nor are the large

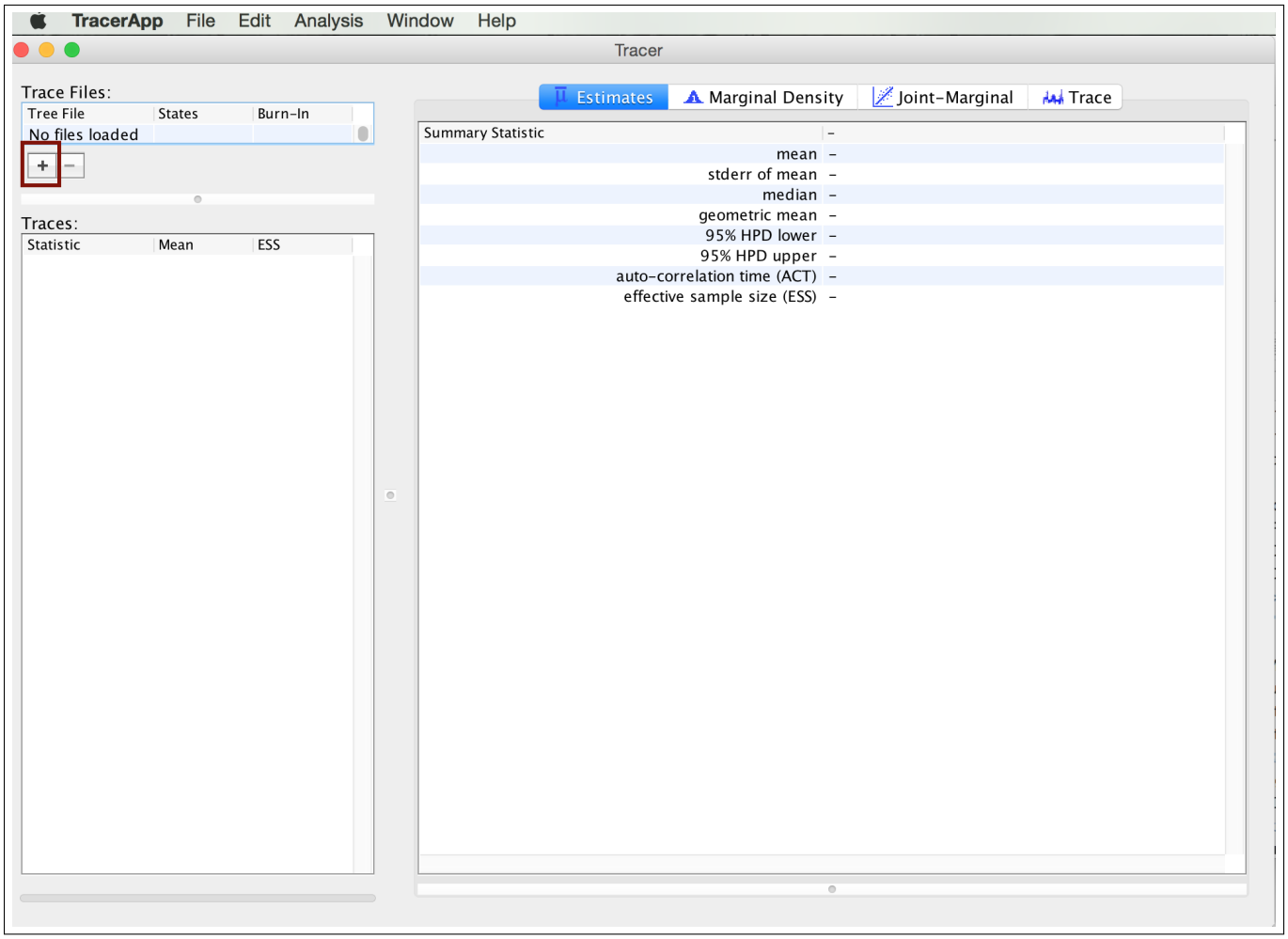

Figure 4: The Tracer window. To add data, click on the "+" sign, highlighted in red above.

square-shaped steps in the trace. The presence of a trend, or of large leaps in a parameter value might indicate that your MCMC is not mixing well. You can read more about MCMC tuning and improving mixing in the tutorial RB_MCMC_Intro_Tutorial and RB_MCMC_Tutorial.

Are there any parameters in your log files that show trends or large leaps? What steps might you take to solve these issues?

3.11.2 Summarize Tree

Start up RevBayes at the command line. You should do this from within the `RB_TotalEvidenceDating_FBD_Tutorial` directory. We will use functions implemented in RevBayes to summarize our trees.

```
./rb
```

Read in the MCMC sample of trees from file. The first thing we need to do is remove taxa for which we

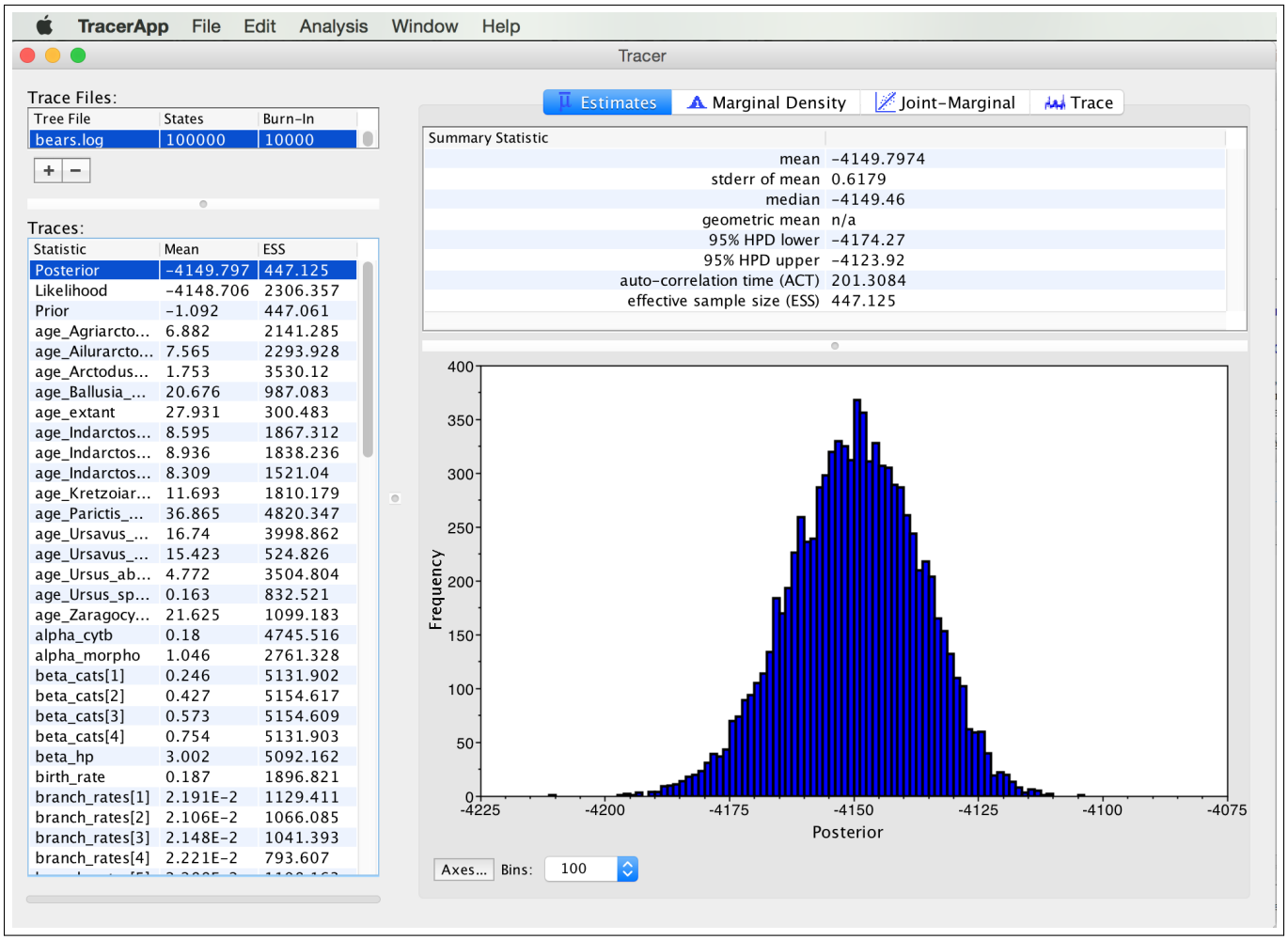

Figure 5: The Estimates window. The left-hand window provides mean and ESS of the chain. The right-hand window visualizes the distribution of samples.

did not have any molecular or morphological data. These taxa haven't been placed on the tree due to their actual phylogenetic relationships, so it would be misleading to include them in a summary tree. We will use the **fnPruneTree** function to remove them. Then we will use the mccTree function to return a maximum clade credibility tree. The MCC tree is the tree with the maximum product of the posterior clade probabilities.

```
trace = readTreeTrace("output/bears.trees")

for(i in 1:trace.size())
{
    trees[i] = fnPruneTree(trace.getTree(i), pruneTaxa=v(taxa[17],taxa[20]))
}

trace_pruned = treeTrace(trees)
mccTree(trace_pruned, "output/bears.mcc.tre" )
```

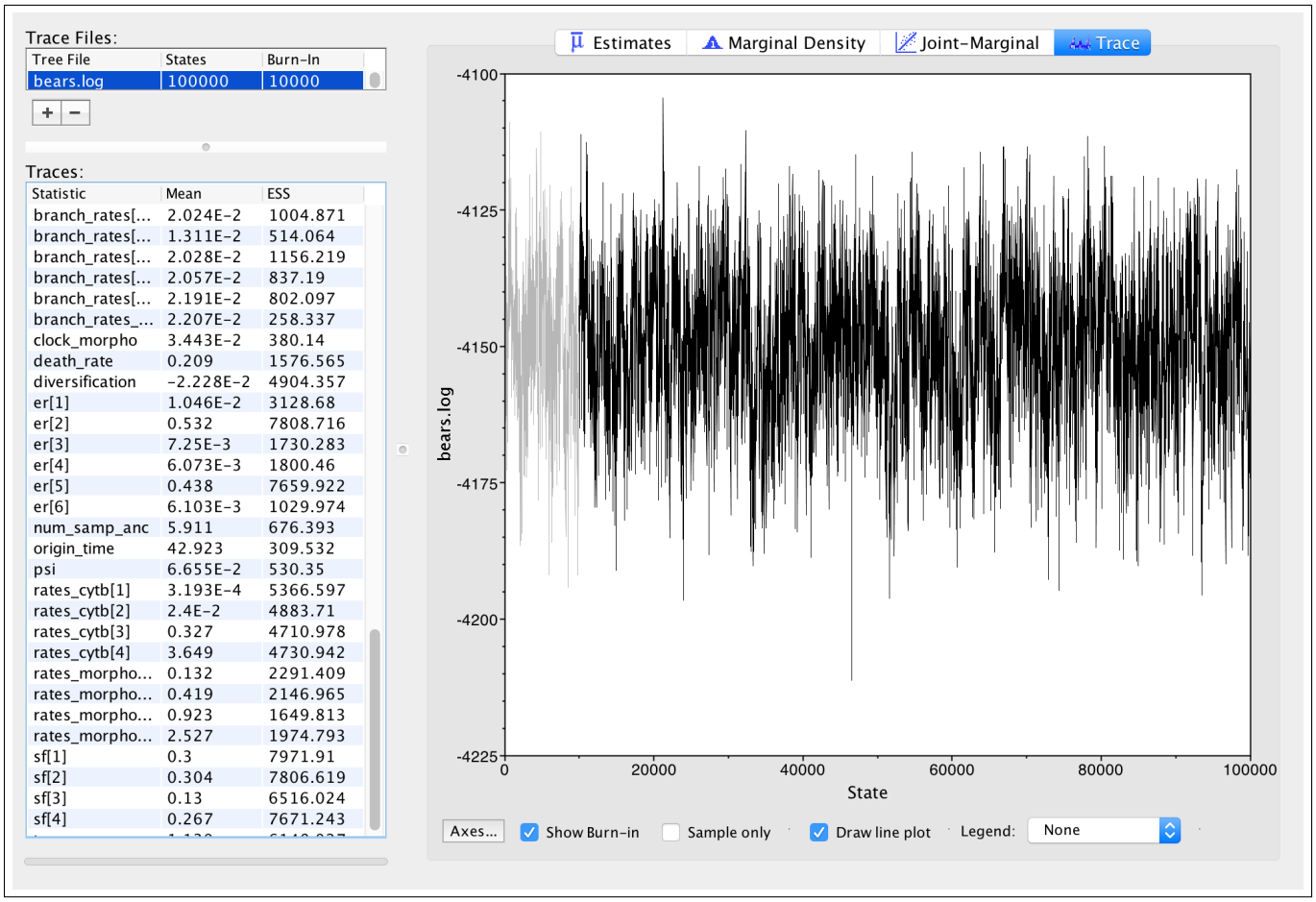

Figure 6: The trace window. The Y-axis is an estimate of the parameter being sampled (in this case likelihood score) and the X-axis is position in the MCMC chain.

There are sampled ancestors present in the tree. This makes visualizing the tree fairly difficult in traditional tree viewers. Today, we will make use of a browser-based tree viewer called IcyTree. This tree viewer has the ability to view sampled ancestor trees, as well as many other nice abilities for tree visualization. Navigate to the page and open the tree.

Try to replicate Fig. 7. Hint: Look under style, at 'Mark Singletons'. Why might a node with a sampled ancestor be referred to as a singleton? Try clicking on a branch. What are the fields telling you? Are the ages of the fossil tips the same as what was shown in Tracer?

References

Ronquist, F., S. Klopfstein, L. Vilhelmsen, S. Schulmeister, D. L. Murray, and A. P. Rasnitsyn. 2012. A total-evidence approach to dating with fossils, applied to the early radiation of the Hymenoptera. Systematic Biology 61:973–999.

Zuckerkandl, E. and L. Pauling. 1962. Molecular disease, evolution, and genetic heterogeneity. Pages 189–225 *in* Horizons in Biochemistry (M. Kasha and B. Pullman, eds.) Academic Press, New York.

Version dated: December 23, 2016

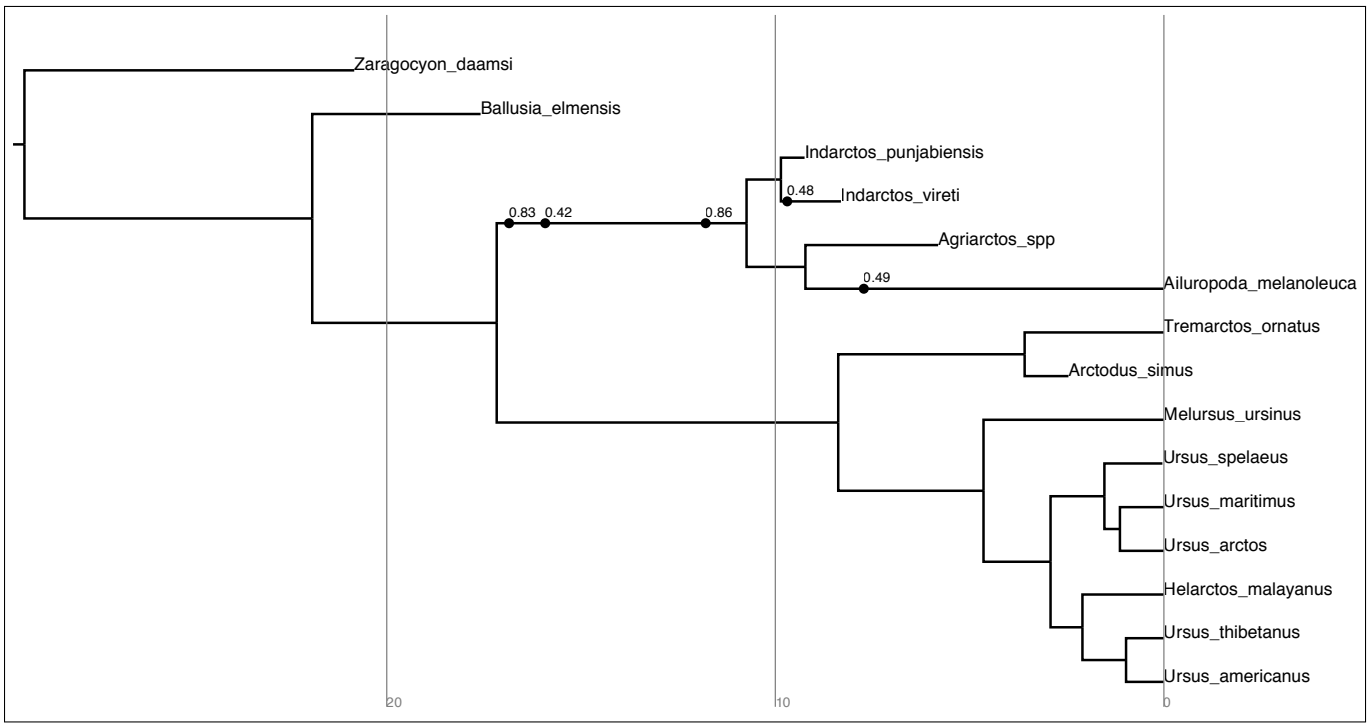

Figure 7: This is a place-holder figure.