Phylogenetic Inference using `RevBayes`

Discrete Morphology

April M. Wright and Michael J. Landis

1 Introduction

While molecular data have become the default for building phylogenetic trees for many types of evolutionary analysis, morphological data remains important, particularly for analyses involving fossils. The use of morphological data raises special considerations for model-based methods for phylogenetic inference. Morphological data are typically collected to maximize the number of parsimony-informative characters - that is, the characters that provide information in favor of one topology over another. Morphological characters also do not carry common meanings from one character in a matrix to the next; character codings are made arbitrarily. These two factors require extensions to our existing phylogenetic models. Accounting for the complexity of morphological characters remains challenging. This tutorial will provide a discussion of modeling morphological characters, and will demonstrate how to perform Bayesian phylogenetic analysis with morphology using `RevBayes` (**?**).

1.1 Contents

The Discrete Morphology guide contains several tutorials

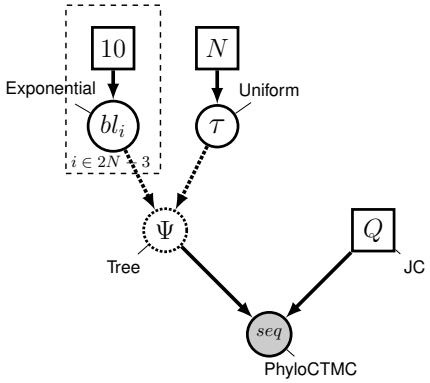
- Section 2: Overview of the Discrete Morphological models
- Section 3: A simple discrete morphology analysis
- Section 4: A model for allowing state frequency variation across binary characters
- Section 5: A model for allowing state frequency variation across binary and multistate characters
- Section 6: Evaluating the MCMC

1.2 Recommended tutorials

The Discrete Morphology tutorials assume the reader is familiar with the content covered in the following `RevBayes` tutorials

- **`Rev` Basics**
- **Molecular Models of Character Evolution**
- **Running and Diagnosing an MCMC Analysis**
- **Divergence Time Estimation and Node Calibrations**

2 Overview of Discrete Morphology Models




```
for (i in 1:n_branches) {
        bl[i] ~ dnExponential(10.0)
}
topology ~ dnUniformTopology(taxa)
psi := treeAssembly(topology, bl)


Q_morpho <- fnJC(2)

phyMorpho ~ dnPhyloCTMC( tree=phylogeny,
siteRates=rates_morpho, Q=Q_morpho,
type="Standard", coding="variable" )

phyMorpho.clamp( data )
```

Figure 1: Graphical model showing the Mk model (left panel). Rev code specifying the Mk model is on the right-hand panel.

Molecular data forms the basis of most phylogenetic analyses today. However, morphological characters remain relevant: Fossils often provide our only direct observation of extinct biodiversity; DNA degradation can make it difficult or impossible to obtain sufficient molecular data from fragile museum specimens. Using morphological data can help researchers include specimens in their phylogeny that might be left out of a molecular tree.

To understand how morphological characters are modeled, it is important to understand how characters are collected. Unlike in molecular data, for which homology is algorithmically determined, homology in a character is typically assessed by an expert. Biologists will typically decide what characters are homologous by looking across specimens at the same structure in multiple taxa; they may also look at the developmental origin of structures in making this assessment (**?**). Once homology is determined, characters are broken down into states, or different forms a single character can take. The state '0' commonly refers to absence, meaning that character is not present. In some codings, absence will mean that character has not evolved in that group. In others, absence means that that character has not evolved in that group, and/or that that character has been lost in that group (**?**). This type of coding is arbitrary, but both **non-random** and **meaningful**, and poses challenges for how we model the data.

Historically, most phylogenetic analyses using morphological characters have been performed using the maximum parsimony optimality criterion. Maximum parsimony analysis involves proposing trees from the morphological data. Each tree is evaluated according to how many changes it implied in the data, and the tree that requires the fewest changes is preferred. In this way of estimating a tree, a character that does not change, or changes only in one taxon, cannot be used to discriminate between trees (i.e., it does not favor a topology). Therefore, workers with parsimony typically do not collect characters that are parsimony uninformative.

In 2001, Paul Lewis (**?**) introduced a generalization of the Jukes-Cantor model of sequence evolution for use with morphological data. This model, called the Mk (Markov model, assuming each character is in one of k states) model provided a mathematical formulation that could be used to estimate trees from morphological data in both likelihood and Bayesian frameworks. While this model is a useful step forward, as a generalization of the Jukes-Cantor, it still makes fairly simplistic assumptions. This tutorial will guide you through estimating a phylogeny with the Mk model, and two useful extensions to the model.

2.1 The Mk Model

The Mk model is a generalization of the Jukes-Cantor model of nucleotide sequence evolution, which we discussed in **Molecular Models of Character Evolution**. The Q matrix for a two-state Mk model looks like so:

$$Q = \begin{pmatrix} -\mu_0 & \mu_{01} \\ \mu_{10} & -\mu_1 \end{pmatrix} ,$$

This matrix can be expanded to accommodate multi-state data, as well:

$$Q = \begin{pmatrix} -\mu_0 & \mu_{01} & \mu_{02} & \mu_{03} \\ \mu_{10} & -\mu_1 & \mu_{12} & \mu_{13} \\ \mu_{20} & \mu_{21} & -\mu_2 & \mu_{23} \\ \mu_{30} & \mu_{31} & \mu_{32} & -\mu_3 \end{pmatrix} ,$$

However, the Mk model sets transitions to be equal from any state to any other state. In that sense, our multistate matrix really looks like this:

$$Q = \begin{pmatrix} -(k-1)\mu & \mu & \mu & \mu \\ \mu & -(k-1)\mu & \mu & \mu \\ \mu & \mu & -(k-1)\mu & \mu \\ \mu & \mu & \mu & -(k-1)\mu \end{pmatrix} ,$$

Because this is a Jukes-Cantor-like model (**?**), state frequencies do not vary as a model parameter. These assumptions may seem unrealistic. However, all models are a compromise between reality and generalizability. Prior work has demonstrated that, in many conditions, the model does perform adequately (**?**). Because morphological characters do not carry common meaning across sites in a matrix in the way that nucleotide characters do, making assumptions that fit all characters is challenging. A visualization of this simple model can be seen in Fig. 2.

We will first perform a phylogenetic analysis using the Mk model. In further sections, we will explore how to relax key assumptions of the Mk model.

2.2 Ascertainment Bias

When Lewis first introduced the Mk model, he observed that branch lengths on the trees were greatly inflated. The reason for this is that when morphological characters are collected, characters that do not vary, or vary in a non-parsimony-informative way (such as autapomorphies) are excluded. Excluding these low-rate characters causes the overall amount of evolution to be over-estimated. This causes an inflation in the branch lengths **?**.

Therefore, when performing a morphological phylogenetic analysis, it is important to correct for this bias. There are numerous statistically valid ways to perform this correction **?**. Original corrections simulated invariant and non-parsimony informative characters along the proposed tree. The likelihood of these characters would then be calculated and used to normalize the total likelihood value. `RevBayes` implements a dynamic programming approach that calculates the same likelihood, but does so faster.

3 Example: Inferring a Phylogeny of Fossil Bears Using the Mk Model

In this example, we will use morphological character data from 18 taxa of extinct bears (**?**). The dataset contains 62 binary characters, a fairly typical dataset size for morphological characters.

3.1 Tutorial Format

This tutorial follows a specific format for issuing instructions and information.

> The boxed instructions guide you to complete tasks that are not part of the `RevBayes` syntax, but rather direct you to create directories or files or similar.

Information describing the commands and instructions will be written in paragraph-form before or after they are issued.

All command-line text, including all `Rev` syntax, are given in **`monotype font`**. Furthermore, blocks of `Rev` code that are needed to build the model, specify the analysis, or execute the run are given in separate shaded boxes. For example, we will instruct you to create a constant node called **`example`** that is equal to `1.0` using the `<-` operator like this:

```
example <- 1.0
```

It is important to be aware that some PDF viewers may render some characters given as `Rev commands` differently. Thus, if you copy and paste text from this PDF, you may introduce some incorrect characters. Because of this, we recommend that you type the instructions in this tutorial or copy them from the scripts provided.

3.2 Data and Files

> On your own computer, create a directory called `RB_DiscreteMorphology_Tutorial` (or any name you like).
>
> In this directory download and unzip the archive containing the data files: `data.zip`.
>
> This will create a folder called **`data`** that contains the files necessary to complete this exercise.

3.3 Getting Started

> Create a new directory (in **`RB_DiscreteMorphology_Tutorial`**) called `scripts`. (If you do not have this folder, please refer to the directions in section 3.2.)

When you execute RevBayes in this exercise, you will do so within the main directory you created (**RB_DiscreteMorphology_Tutorial**), thus, if you are using a Unix-based operating system, we recommend that you add the RevBayes binary to your path.

3.4 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[1] and create a set of modular files that will be easily managed and interchanged. In this first section, you will write the following files from scratch and save them in the **scripts** directory:

- **mcmc_mk.Rev**: the master Rev file that loads the data, the separate model files, and specifies the monitors and MCMC sampler.
- **model_mk.Rev**: specifies the model describing discrete morphological character change (binary characters).

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.

> Open your text editor and create the master Rev file called mcmc_Mk.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_Mk.Rev** and complete the Rev script with the instructions given in section 3.6.

3.4.1 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. Import the morphological character matrix and assign it to the variable **morpho**.

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

3.4.2 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.

[1] In section **??** we offer a recommendation for a text editor.

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

Create a new constant node called **n_taxa** that is equal to the number of species in our analysis (18). We will also create a constant node of the taxon names. This list will be used to initialize the tree.

```
taxa <- morpho.names()
n_taxa <- morpho.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_Mk.Rev** in the **scripts** directory.

We will now move on to the next Rev file and will complete **mcmc_Mk.Rev** in section 3.6.

3.5 The Mk Model

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

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

First, we will create a vector of moves on branch lengths. This should be familiar from the **RB_CTMC** tutorial:

```
br_len_lambda ~ dnExp(0.2)
moves[mvi++] = mvScale(br_len_lambda, weight=5)
nbr <- 2*names.size() - 3
for (i in 1:nbr){
    br_lens[i] ~ dnExponential(br_len_lambda)
    moves[mvi++] = mvScale(br_lens[i])
}
```

Next, we will create a Q matrix. Recall that the Mk model is simply a generalization of the JC model. Therefore, we will create a 2x2 Q matrix using **fnJC**, which initializes Q-matrices with equal transition probabilities between all states.

```
Q_morpho <- fnJC(2)
```

Now that we have the basics of the model specified, we will add Gamma-distributed rate variation and specify moves on the parameter to the Gamma distribution.

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

#Moves on the parameters to the Gamma distribution.
moves[mvi++] = mvScale(alpha_morpho, lambda=0.01, weight=5.0)
moves[mvi++] = mvScale(alpha_morpho, lambda=0.1, weight=3.0)
moves[mvi++] = mvScale(alpha_morpho, lambda=1, weight=1.0)
```

Next we assemble the tree and specify a move on the topology.

```
tau ~ dnUniformTopology(names)
phylogeny := treeAssembly(tau, br_lens)
moves[mvi++] = mvNNI(tau, weight=2*nbr)
moves[mvi++] = mvSPR(tau, weight=nbr)
tree_length := phylogeny.treeLength()
```

Lastly, we set up the CTMC. This should be familiar from the **RB_CTMC** tutorial. We see some familiar pieces: tree, Q matrix and site_rates. We also have two new keywords: data type and coding. The data type argument specifies the type of data - in our case, "Standard", the specification for morphology. Coding specifies what type of ascertainment bias is expected. We are using the 'variable' correction, as we have no invariant character in our matrix. If we also lacked parsimony non-informative characters, we would use the coding 'informative'.

```
phyMorpho ~ dnPhyloCTMC(tree=phylogeny, siteRates=rates_morpho, Q=Q_morpho, type="
    Standard", coding="variable")
phyMorpho.clamp(morpho)
```

All of the components of the model are now specified.

3.6 Complete Master Rev File

Return to the master Rev file called mcmc_Mk.Rev in the **scripts** directory.

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

3.6.1 Source Model Scripts

RevBayes uses the **source()** function to load commands from Rev files into the workspace. Use this function to load in the model scripts we have written in the text editor and saved in the **scripts** directory.

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

3.6.2 Create Model Object

We can now create our workspace model variable with our fully specified model DAG. We will do this with the **model()** function and provide a single node in the graph (**phylogeny**).

```
mymodel = model(phylogeny)
```

The object **mymodel** is a wrapper around the entire model graph and allows us to pass the model to various functions that are specific to our MCMC analysis.

3.6.3 Specify Monitors and Output Filenames

The next important step for our master Rev file is to specify the monitors and output file names. For this, we create a vector called **monitors** that will each sample and record or output our MCMC.

First, we will specify a workspace variable to iterate over the **monitors** vector.

```
mni = 1
```

The first monitor we will create will monitor every named random variable in our model graph. This will include every stochastic and deterministic node using the **mnModel** monitor. The only parameter that is not included in the **mnModel** is the tree topology. Therefore, the parameters in the file written by this monitor are all numerical parameters written to a tab-separated text file that can be opened by accessory programs for evaluating such parameters. We will also name the output file for this monitor and indicate that we wish to sample our MCMC every 10 cycles.

```
monitors[mni++] = mnModel(filename="output/mk_simple.log", printgen=10)
```

The **mnFile** monitor writes any parameter we specify to file. Thus, if we only cared about the branch lengths and nothing else (this is not a typical or recommended attitude for an analysis this complex) we wouldn't use the **mnModel** monitor above and just use the **mnFile** monitor to write a smaller and simpler output file. Since the tree topology is not included in the **mnModel** monitor (because it is not numerical), we will use **mnFile** to write the tree to file by specifying our **phylogeny** variable in the arguments.

```
monitors[mni++] = mnFile(filename="output/mk_simple.trees", printgen=10, phylogeny)
```

The third monitor we will add to our analysis will print information to the screen. Like with **mnFile** we must tell **mnScreen** which parameters we'd like to see updated on the screen.

```
monitors[mni++] = mnScreen(printgen=10)
```

Finally, we'll create an ancestral state monitor, which is described in more detail in Section **??**.

```
monitors[mni++] = mnJointConditionalAncestralState(
    tree=phylogeny,
    ctmc=phyMorpho,
    filename="output/mk_simple.states.txt",
    type="Standard",
    printgen=10,
    withStartStates=false)
```

3.6.4 Set-Up the MCMC

Once we have set up our model, moves, and monitors, we can now create the workspace variable that defines our MCMC run. We do this using the **mcmc()** function that simply takes the three main analysis components as arguments.

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

The MCMC object that we named **mymcmc** has a member method called **.run()**. This will execute our analysis and we will set the chain length to **10000** cycles using the **generations** option.

```
mymcmc.run(generations=20000)
```

Once our Markov chain has terminated, we will want **RevBayes** to close. Tell the program to quit using the **q()** function.

```
q()
```

You made it! Save all of your files.

3.7 Execute the MCMC Analysis

With all the parameters specified and all analysis components in place, you are now ready to run your analysis. The `Rev` scripts you just created will all be used by `RevBayes` and loaded in the appropriate order.

Begin by running the `RevBayes` executable. In Unix systems, type the following in your terminal (if the `RevBayes` binary is in your path):

```
rb
```

Provided that you started `RevBayes` from the correct directory (**`RB_DiscreteMorphology_Tutorial`**), you can then use the **`source()`** function to feed `RevBayes` your master script file (**`mcmc_mk.Rev`**).

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

This will execute the analysis and you should see the following output (though not the exact same values):

```
source("scripts/mcmc_mk.rev")
  Processing file "scripts/mcmc_mk.rev"
  Successfully read one character matrix from
file 'data/bears.nex'
  Processing file "scripts/mk_simple.rev"
  Processing of file "scripts/mk_simple.rev"
completed

  Running MCMC simulation
  This simulation runs 1 independent replicate.
  The simulator uses 37 different moves in a
random move schedule with 37 moves per iteration

Iter      |     Posterior  |    Likelihood
|        Prior  |   elapsed  |       ETA


-----------------------------------------------------------------------------------------------
0         |      -685.779  |      -666.105  |      -19.6731  |  00:00:00  |  --:--:--  |

10        |      -633.737  |      -608.951  |       -24.786  |  00:00:00  |  --:--:--  |

20        |      -593.634  |      -558.797  |      -34.8368  |  00:00:00  |  00:00:00  |
30        |      -578.661  |      -536.125  |      -42.5362  |  00:00:00  |  00:00:00  |
40        |       -544.22  |      -514.098  |      -30.1223  |  00:00:00  |  00:00:00  |
50        |      -515.505  |      -492.988  |      -22.5169  |  00:00:00  |  00:00:00  |
60        |      -483.695  |      -461.347  |      -22.3479  |  00:00:00  |  00:00:00  |

...
```

When the analysis is complete, `RevBayes` will quit and you will have a new directory called **`output`** that will contain all of the files you specified with the monitors (Sect. 3.6.3).

3.7.1 Ascertainment Bias

As discussed in (Sect. 2.2), we also need to correct for ascertainment bias. Once your initial Mk model estimation, source the mcmc_simple.Rev script. This file is an estimation of the Mk model without any correction for ascertainment bias. We will use this for comparison in a moment.

4 Example: Relaxing the Assumption of Equal Transition Probabilities

Make a copy of the MCMC and model files you made earlier. Call them **mcmc_mk_dicretized.Rev** and **model_mk_discretized.Rev**. These will contain the new model parameters and models.

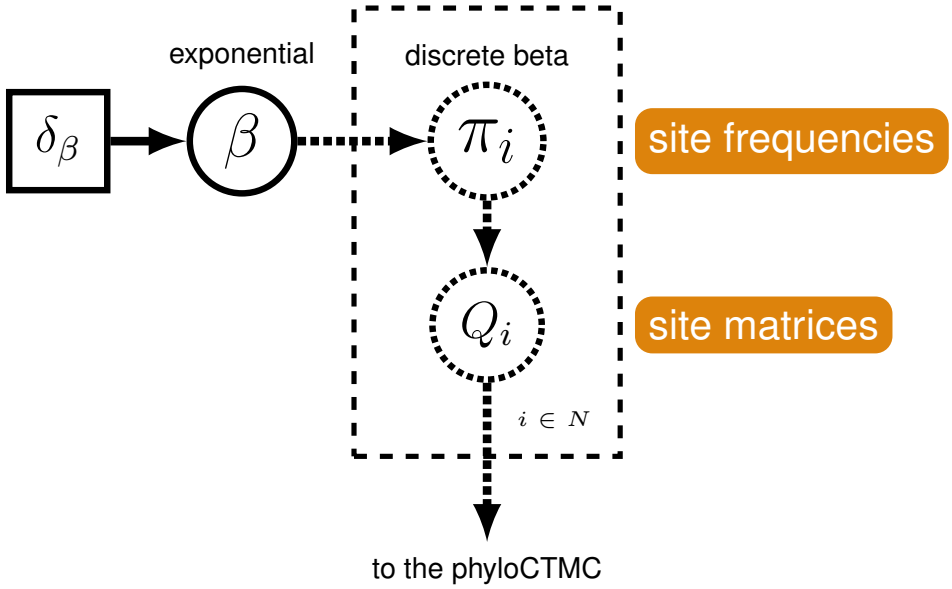


Figure 2: Graphical model demonstrating the discretized Beta distribution for allowing variable state frequencies.

The Mk model makes a number of assumptions, but one that may strike you as unrealistic is the assumption that characters are equally likely to change from any one state to any other state. That means that a trait is as likely to be gained as lost. While this may hold true for some traits, we expect that it may be untrue for many others.

RevBayes has functionality to allow us to relax this assumption. We do this by specifying a Beta prior on state frequencies. Remember from the **RB_CTMC** lesson that stationary frequencies impact how likely we are to see changes in a character. For example, it may be very likely, in a character, to change from 0 to 1. But if the frequency of 0 is very low, we will still seldom see this change.

We can exploit the relationship between state frequencies and observed changes to allow for variable Q matrices across characters (Fig. 2). To do this, we generate a Beta distribution on state frequencies, and use the state frequencies from that Beta distribution to generate a series of Q-matrices to use to evaluate our data (**?**).

This type of model is called a **mixture model**. There are assumed to be subdivisions in the data, which may require different parameters (in this case, state frequencies). These subdivisions are not defined *a priori*. This model has previously been shown to be effective for a range of empirical and simulated datasets (**?**).

4.1 Modifying the MCMC File

At each place in which the output files are specified in the MCMC file, change the output path so you don't overwrite the output from the previous exercise. For example, you might call your output file **output/mk_discretized.log** and **output/mk_discretized.trees**. Change source statement to indicate the new model file.

4.2 Modifying the Model File

Open the new model file that you created. We need to modify the way in which the Q matrix is specified. We will use a discretized Beta distribution to place a prior on state frequencies. The Beta distribution has two parameters, α and β. These two parameters specify the shape of the distribution. State frequencies will be evaluated according to this distribution, in the same way that rate variation is evaluated according to the Gamma distribution. The discretized distribution is split into multiple classes, each with it's own set of frequencies for the 0 and 1 characters. The number of classes can vary; we have chosen 4 for tractability.

```
n_cats = 4
alpha_ofbeta ~ dnExponential( 1 )
beta_ofbeta ~ dnExponential( 1 )
moves[mvi++] = mvScale(alpha_ofbeta, lambda=1, weight=1.0 )
moves[mvi++] = mvScale(alpha_ofbeta, lambda=0.1, weight=3.0 )
moves[mvi++] = mvScale(alpha_ofbeta, lambda=0.01, weight=5.0 )
moves[mvi++] = mvScale(beta_ofbeta, lambda=1, weight=1.0 )
moves[mvi++] = mvScale(beta_ofbeta, lambda=0.1, weight=3.0 )
moves[mvi++] = mvScale(beta_ofbeta, lambda=0.01, weight=5.0 )
```

Above, we initialized the number of categories, the parameters to the Beta distribution, and the moves on the parameters to the Beta.

Next, we set the categories to each represent a quadrant of the Beta distribution specified by the **alpha_ofbeta** and **beta_ofbeta**. The **+1** values are added to the beta shape and scale parameters to prevent model over-fitting.

```
cats := fnDiscretizeBeta(alpha_ofbeta+1, beta_ofbeta+1, 4)
```

If you were to print the **cats** variable, you would see a list of state frequencies like so:

```
0.0780943
0.40457
0.769453
0.97106
```

Using these state frequencies, we will generate a new vector of Q matrices. Because we are varying the state frequencies, we must use a Q matrix generation function that allows for state frequencies to vary as a parameter. We will, therefore, use the **fnF81** function.

```
for (i in 1:cats.size())
{
    Q[i] := fnF81(simplex(abs(1-cats[i]), cats[i]))
}
```

Once we've made the our vector of matrices, we specified moves on our matrix vector:

```
matrix_probs ~ dnDirichlet(v(1,1,1,1))
moves[mvi++] = mvSimplexElementScale(matrix_probs, alpha=10, weight=1.0)
```

This Dirichlet prior says that no category is expected to have more characters than another. If you expected some category to hold more of the characters, you could put more weight on that category.

The only other specification that needs to change in the model file is the CTMC:

```
phyMorpho ~ dnPhyloCTMC(tree=phylogeny, siteRates=rates_morpho, Q=Q, type="Standard",
    coding="variable", siteMatrices=matrix_probs)
```

You'll notice that we've added a command to tell the CTMC that we have multiple site matrices that will be applied to different characters in the matrix.

4.2.1 Set-Up the MCMC

The MCMC chain set-up does not need to change. Run the new MCMC file, just as you ran the plain Mk file. This estimation will take longer than the Mk model, due to increased model complexity.

5 Site-Heterogeneous Discrete Morphology Model

In the previous example, we explored allowing among-character variation in state frequencies. This is an excellent start for allowing more complex models for morphology. But this approach also has several shortcomings. First, because we use a Beta distribution, this model really only works for binary data. Secondly, oftentimes, we will not have a good idea of the shape of the distribution from which we expect state frequencies to be drawn.

To accommodate for these concerns, `RevBayes` also has a model that is similar to the CAT model (**?**).

The site-heterogeneous discrete morphology model (SHDM) uses a hyperprior on the prior on state frequencies to mix over different possible combinations state frequencies. In this mixture model, F81 Q-matrices (an extension of the Jukes-Cantor which allows for different state frequencies between characters) is initialized from a set of state frequencies. The number of Q-matrices initialized is equal to the number of user-defined categories, as in the discretized Beta model. The state frequencies used to initialize the Q-matrices are drawn from a Dirichelet prior distribution, which is generated by drawing values from an exponential hyperprior distribution. This model is visualized in Fig. 3.

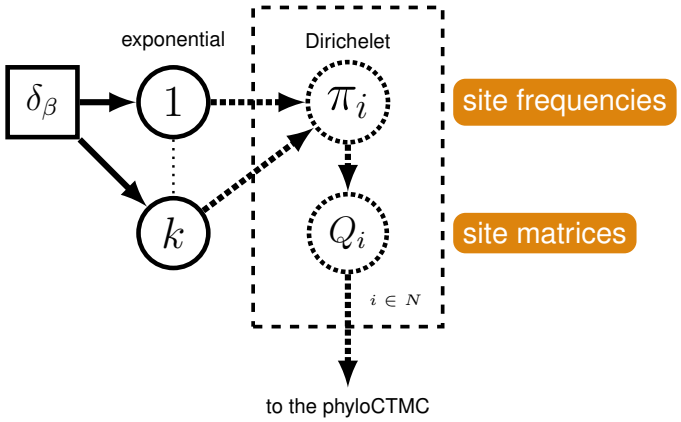


Figure 3: Graphical model demonstrating the Dirichlet prior to allow variable state frequencies in both binary and multistate data.

5.1 Example: Site-Heterogeneous Discrete Morphology Model

Make a copy of the MCMC and model files you just made. Call them **mcmc_mk_hyperprior.Rev** and **model_mk_hyperprior.Rev**. These will contain the new model parameters and models.

5.2 Modifying the MCMC File

At each place in which the output files are specified in the MCMC file, change the output path so you don't overwrite the output from the previous exercise. For example, you might call your output file **output/mk_hyperprior.log** and **output/mk_hyperprior.trees**. We will also monitor Q_morpho and pi. Add Q_morpho and pi to the **mnScreen**. Change source statement to indicate the new model file.

5.3 Modifying the Model File

Open the new model file that you created. We need to modify the way in which the Q-matrix is specified. First, we will create a hyperprior called **dir_alpha** and specify a move on it.

```
dir_alpha ~ dnExponential(1)
moves[mvi++] = mvScale(dir_alpha, lambda=1, weight=1.0 )
moves[mvi++] = mvScale(dir_alpha, lambda=0.1, weight=3.0 )
moves[mvi++] = mvScale(dir_alpha, lambda=0.01, weight=5.0 )
```

This hyperparameter, dir_alpha, will be used as a parameter to a Dirichelet distribution from which our state frequencies will be drawn.

```
pi_prior := v(dir_alpha,dir_alpha)
```

If you were using multistate data, the dir_alpha can be repeated for each state. Next, we will modify our previous loop to use these state frequencies to initialize our Q-matrices.

```
{
        pi[i] ~ dnDirichlet(pi_prior)
    moves[mvi++] = mvSimplexElementScale(pi[i], alpha=10, weight=1.0)

    Q_morpho[i] := fnF81(pi[i])
}
```

In the above loop, for each of our categories, we make a new draw of state frequencies from our Dirichelet distribution (the shape of which is determined by our dir_alpha values). We then use **fnF81** to make our Q-matrices. For each **RevBayes** iteration, we will have 4 pi values and 4 Q-matrices, one for each of the number of categories we specified.

No other aspects of the model file need to change. Run the MCMC as before.

6 Evaluate and Summarize Your Results

6.1 Evaluate MCMC

We will use **Tracer** to evaluate the MCMC samples from our three estimations. Load all three of the MCMC logs into the **Tracer** window. The MCMC chains will not have converged because they have not been run very long. Highlight all three files in the upper left-hand viewer (Fig. 4) by right- or command-clicking all three files.

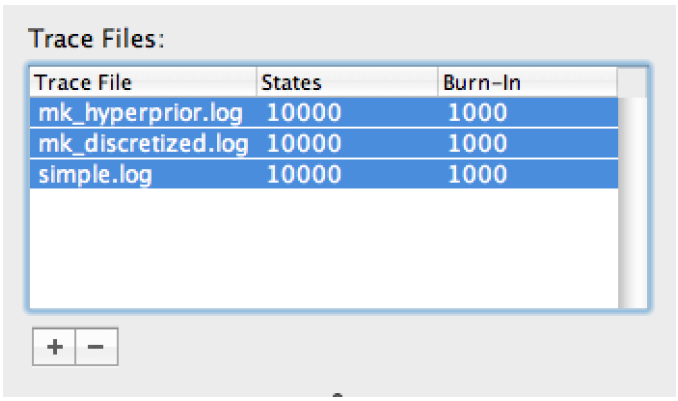


Figure 4: Highlight all three files for model comparison.

Once all three trace logs are loaded and highlighted, first look at the estimated marginal likelihoods. You will notice that the Mk model, as originally proposed by (**?**) is improved by allowing any state frequency heterogeneity at all. The discretized model and the Dirichlet model both represent improvements, but are fairly close in likelihood score to each other (Fig. 5). Likely, we would need to perform stepping stone model assessment to truly tell if the more complicated model is statistically justified. This analysis is too complicated and time-consuming for this tutorial period, but you will find instructions below for performing the analysis.

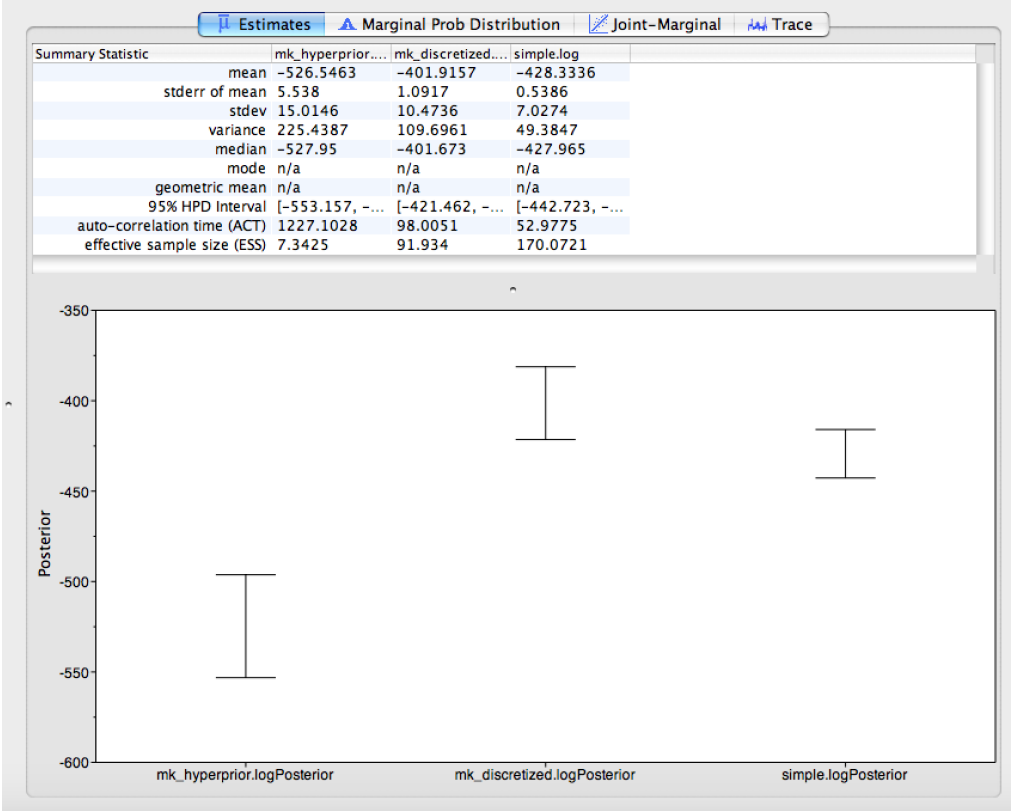


Figure 5: Comparison of likelihood scores for all three models.

Click on the 'Trace' panel. In the lower left hand corner, you will notice an option to color each trace by the file it came from. Choose this option (you may need to expand the window slightly to see it). Next to this option, you can also see an option to add a legend to your trace window. The results of this coloring can be seen in Fig. 6. When the coloring is working, you will see that the Mk model mixes quite well, but that mixing becomes worse as we relax the assumption of equal state frequencies. This is because we are greatly increasing model complexity. Therefore, we would need to run the MCMC chains longer if we were to use these analyses in a paper.

We are interested in two aspects of the posterior distribution. First, all analyses correct for the biased sampling of variable characters except for the `simple` analysis. Then, we expect the `tree_length` variable to be greater for `simple` than for the remaining analyses, because our data are enriched for variation. Figure 7 shows that `tree_length` is approximately 30% greater for `simple` than for `mk_simple`, which are identical except that `mk_simple` corrects for sampling bias. To compare these densities, click the "Marginal Prob Distribution" tab in the upper part of the window, highlight all of the loaded Trace Files, then select `tree_length` from the list of Traces.

Second, we are interested in characterizing the degree of heterogeneity estimated by the beta-discretized model. If the data were distributed by a single morphological rate matrix, then we would expect to see very little variation among the different values in `cats`, and very large values for the shape and scale parameters of the discrete-beta distribution. For example, if `alpha_ofbeta = beta_ofbeta = 1000`, then that would cause all discrete-beta categories to have values approaching 0.5, which approximates a symmetric Mk model.

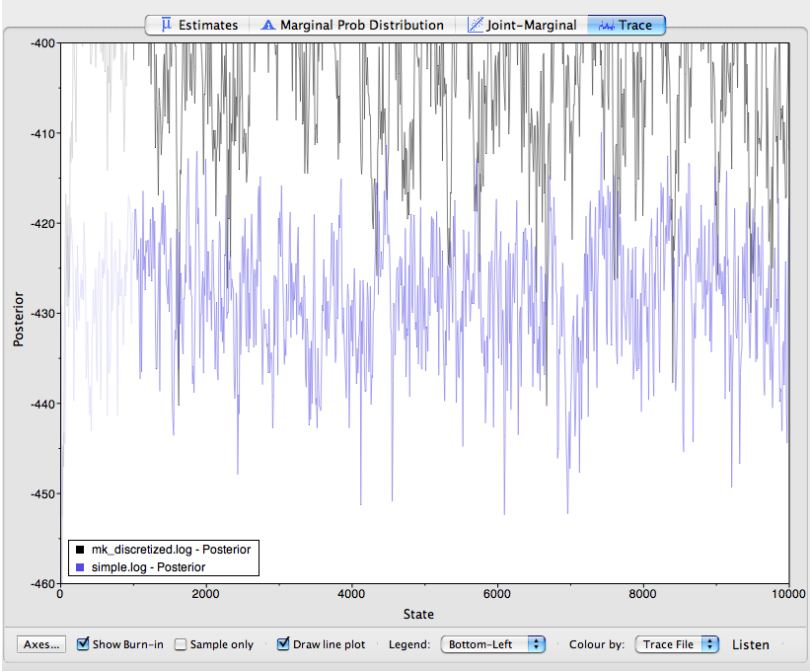


Figure 6: The Trace window. The traces are colored by which version of the Mk model they correspond to.

Figure 8 shows that the four discrete-beta state frequencies do not all have the exact same value. In addition, Figure 9 shows that the priors on the discrete-beta distribution are small enough that we expect to see variance among `cat` values. If the data contained no information regarding the distribution of `cat` values, then the posterior estimates for `alpha_ofbeta` and `beta_ofbeta` would resemble the prior.

6.2 Summarizing tree estimates

The morphology trees estimated in 3 and 4 are summarized using a majority rule consensus tree (MRCT). Clades appearing in $p > 0.5$ of posterior samples are resolved in the MRCT, while poorly support clades with $p \leq 0.5$ are shown as unresolved polytomies. Poor phylogenetic resolution might be caused by having too few phylogenetically informative characters, or it might be due to conflicting signals for certain species relationships. Because phylogenetic information is generated through model choice, let's compare our topological estimates across models.

The MRCTs for the simple model with and without the +v correction are very similar to that for the discretized-beta model (Fig. 10). Note that the scale bars for branch lengths differ greatly, indicating that tree length estimates are inflated without the +v correction, just as we saw when comparing the posterior tree length densities. In general, it is important to assess whether your results are sensitive to model assumptions, such as the degree of model complexity, and any mechanistic assumptions that motivate the model's design. In this case, our tree estimate appears to be robust to model complexity.

Version dated: February 19, 2018

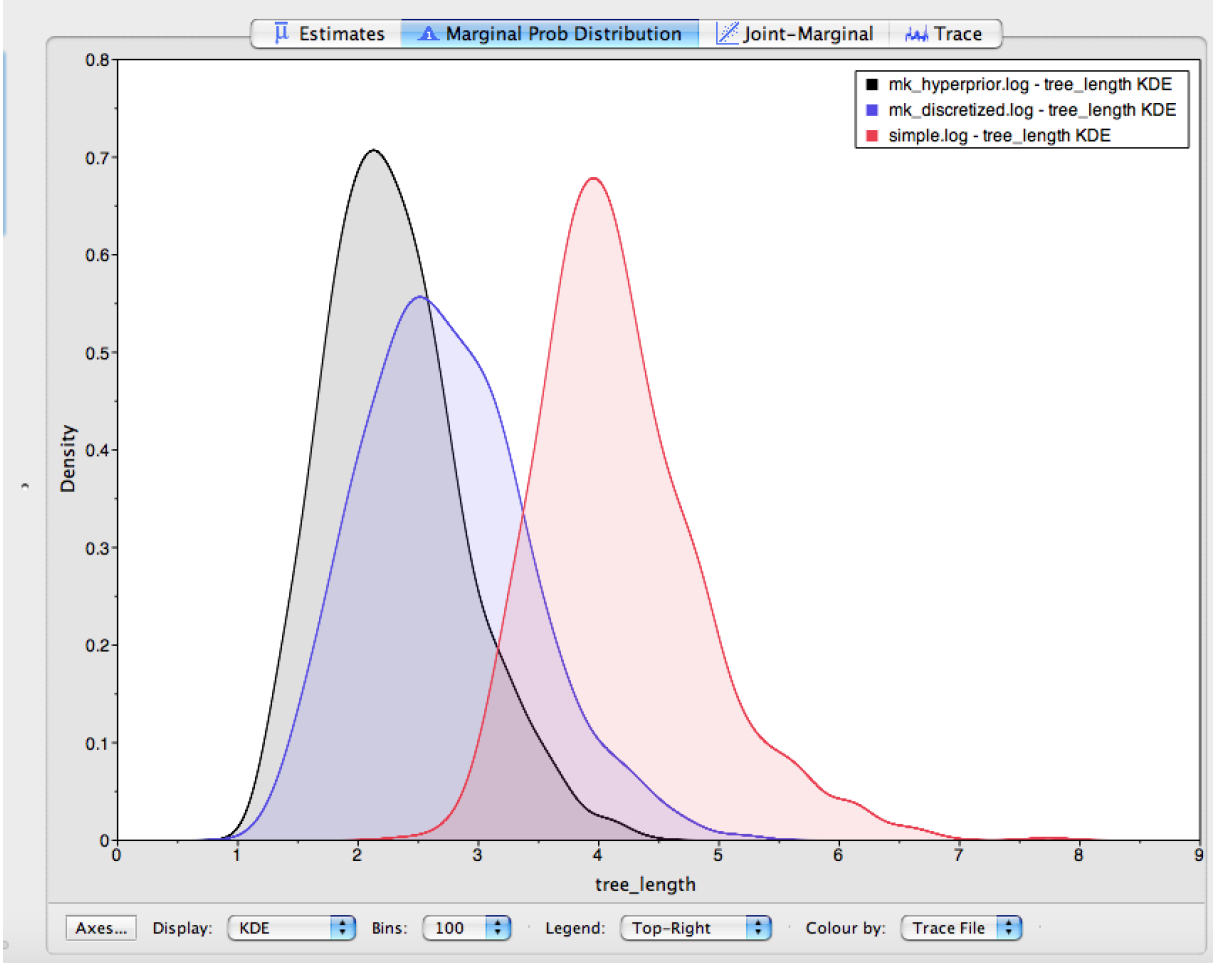


Figure 7: Posterior tree length estimates.

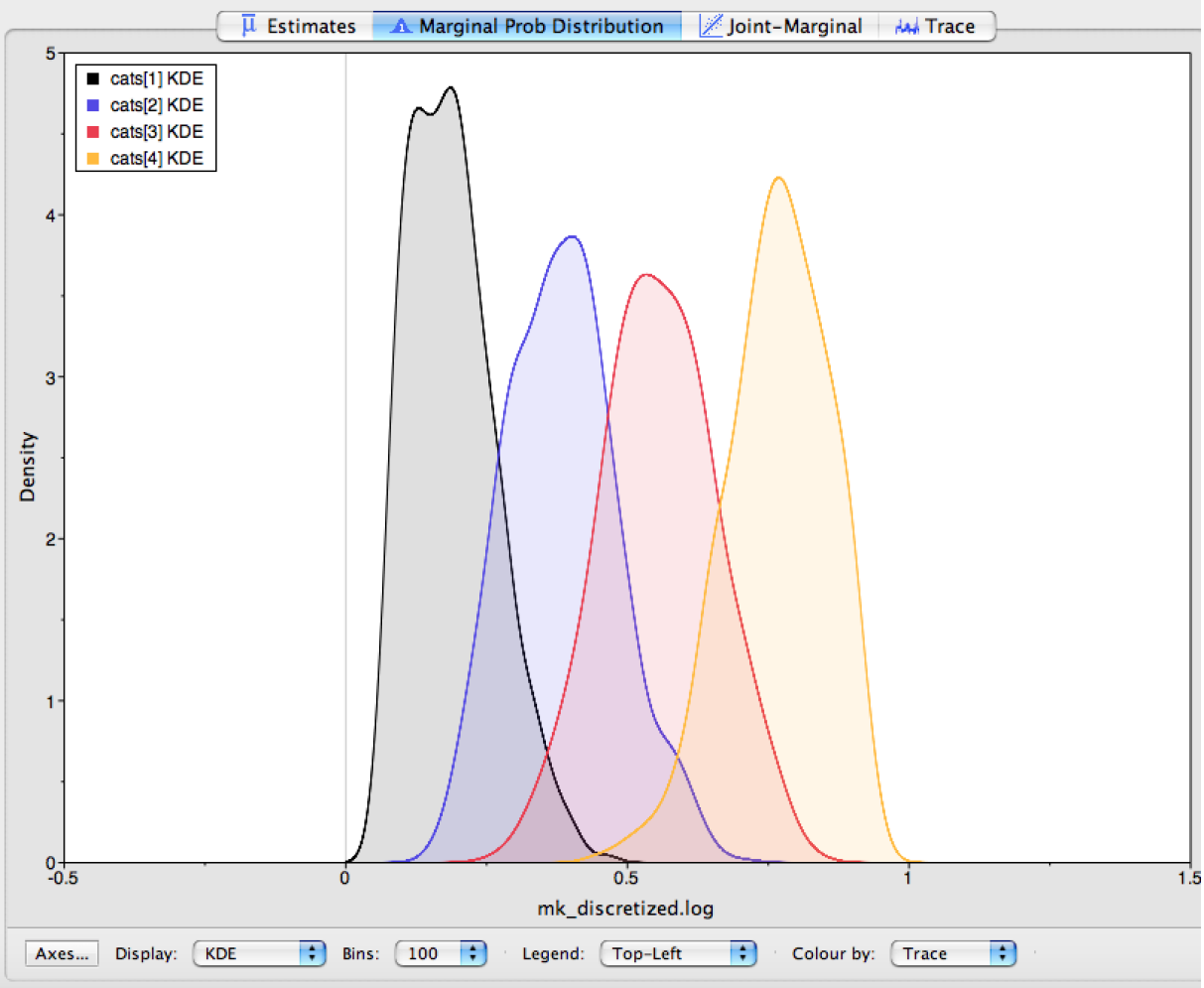


Figure 8: Posterior discretized state frequencies for the discrete-beta model.

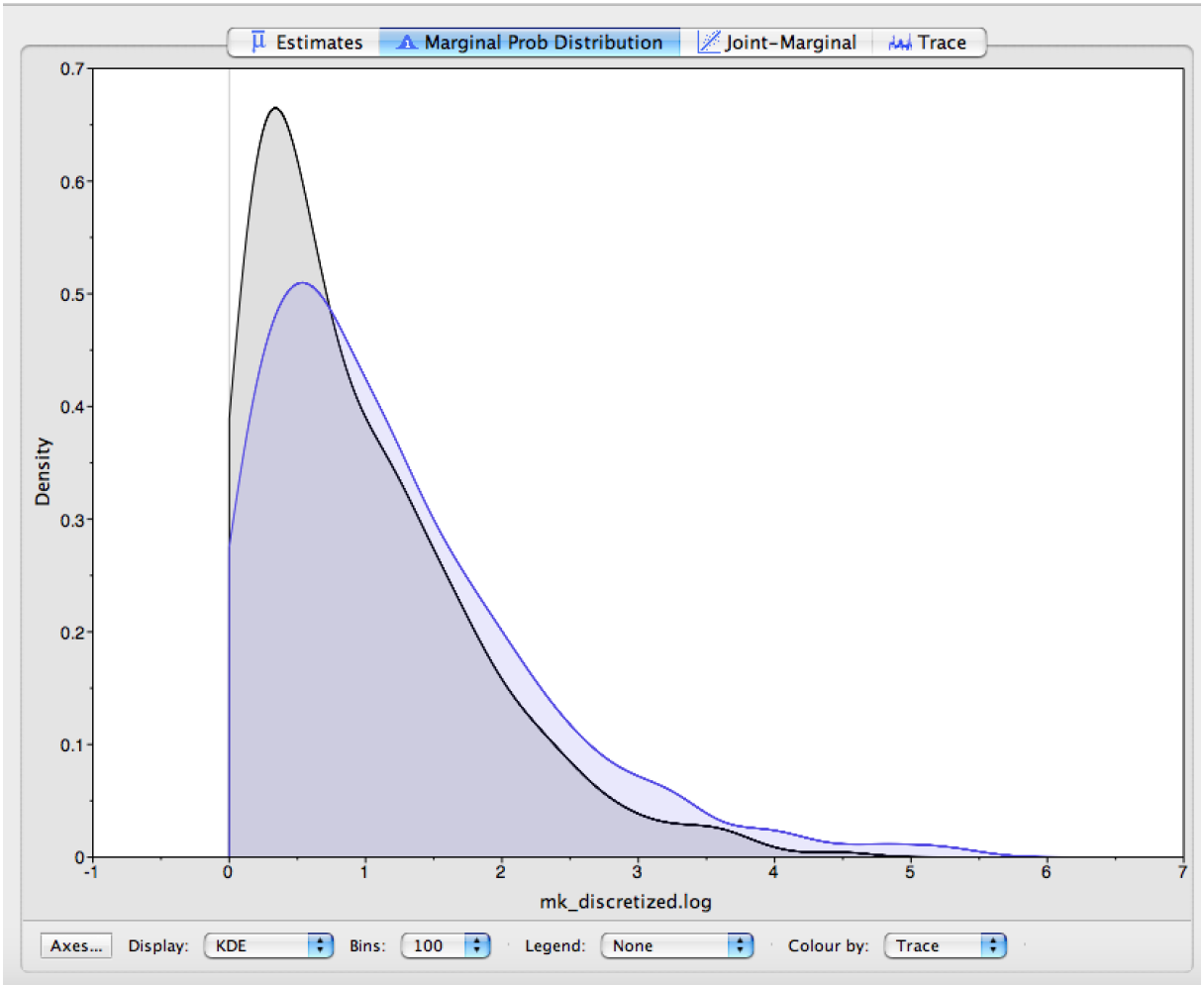


Figure 9: Posterior alpha and beta parameters for the discrete-beta model.

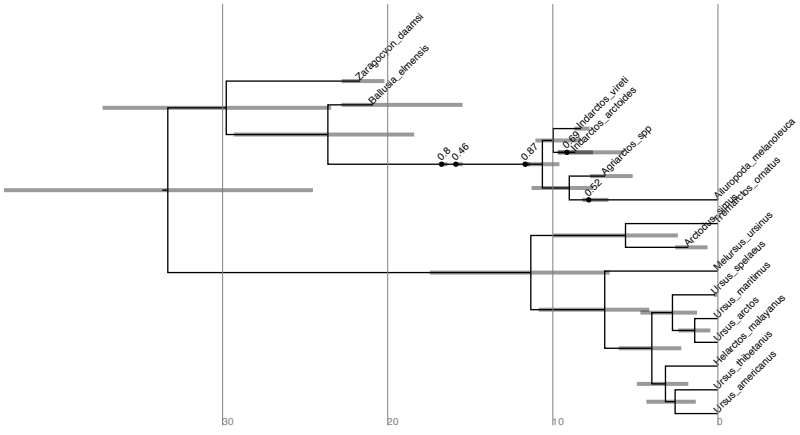


Figure 10: Majority rule consensus tree for the beta-discretized Mkv analysis.