

# Running the old HiSSE function

Jeremy M. Beaulieu

## Getting started

As of version 1.9.6, we now provide a new set of functions that execute more complex and potentially faster version of the HiSSE model described by Beaulieu and O’Meara (2016). One of the main differences here is that the model allows up to 8 hidden categories, which means that both the character-independent model (`hisse.null4`) is called from the same function as the BiSSE and HiSSE models. We also implement a more efficient means of carrying out branch calculations, so it is exceptionally faster than in previous versions. Nevertheless, we have maintained the original implementation of the `hisse` model, and we have kept this tutorial to give some basic information on how to set up and execute the original `hisse` function. For information about how to use the new `hisse()` function see the *A newer, faster HiSSE function* vignette.

Before getting started, be sure to load the `hisse` package:

```
suppressWarnings(library(hisse))
```

```
## Loading required package: ape
## Loading required package: deSolve
## Loading required package: GenSA
## Loading required package: subplex
## Loading required package: nloptr
```

And also be sure to load `diversitree`. We can simulate data using the multistate (MuSSE) model that can be used to create a binary character that has a relation to a “hidden” state:

```
suppressWarnings(library(diversitree))
set.seed(4)
# Essentially we are setting up a model that models the evolution of two binary characters
# Thus, we are assuming the following state combinations 1=00, 2=10, 3=01, 4=11:
pars <- c(0.1,0.1,0.1,0.2, rep(0.03, 4), 0.01,0.01,0,0.01,0,0.01,0.01,0,0.01,0,0.01,0.01)
phy <- tree.musse(pars, max.taxa=50, x0=1, include.extinct=FALSE)
sim.dat <- data.frame(names(phy$tip.state), phy$tip.state)
# Now we want to make the states associated with the second character hidden from us. So,
# we remove states 3 and 4 and make them 1 and 2
sim.dat[sim.dat[,2]==3,2] = 1
sim.dat[sim.dat[,2]==4,2] = 2
# This next step simply forces the character to be binary:
sim.dat[,2] = sim.dat[,2] - 1
```

In the end you have a data set that has a single binary character:

```
head(sim.dat)
```

```
##      names.phy.tip.state. phy.tip.state
## sp3                      sp3          1
## sp10                     sp10          1
## sp12                     sp12          0
## sp17                     sp17          0
## sp18                     sp18          1
## sp19                     sp19          0
```

However, this binary character has actually evolved under a much more complex process than what would be reflected by estimates from a BiSSE model. It is this complexity that HiSSE is trying to estimate.

## Specifying the diversification parameters

Rather than optimizing  $\lambda_i$  and  $\mu_i$  separately, hisse optimizes transformations of these variables. Here we let  $\tau_i = \lambda_i + \mu_i$  define “net turnover”, and we let  $\epsilon_i = \mu_i/\lambda_i$  define the “extinction fraction”. This reparameterization alleviates problems associated with over-fitting when  $\lambda_i$  and  $\mu_i$  are highly correlated, but both matter in explaining the diversity pattern (see discussion of this issue in Beaulieu and O’Meara 2016). The number of free parameters in the model for both net turnover and extinction fraction are specified as index vectors provided to the function call. Each vector contains four entries that correspond to rates associated with the observed states (0 or 1) and the hidden states (A or B). They are ordered as follows: 0A, 1A, 0B, 1B. Note that the index vector can also be set up such that parameters are linked among states or even dropped from the model entirely.

Let’s walk through a couple of examples. Take, for instance, the following index vectors:

```
turnover.anc = c(1,1,0,0)
eps.anc = c(1,1,0,0)
```

In this example, turnover.anc has a single free parameter for both 0A and 1A state combinations; there is a single free parameter for extinction fraction. This is equivalent to a BiSSE model with a fixed turnover and extinction rates across the observed states 0 and 1. Now, say we want to include separate turnover rates for both states 0A and 1A:

```
turnover.anc = c(1,2,0,0)
```

Thus, a full hisse model would thus be,

```
turnover.anc = c(1,2,3,4)
```

which corresponds to four separate net turnover rates for 1=0A, 2=1A, 3=0B, 4=1B. Extinction fraction, needless to say, follows the same format, though including a zero for a state we want to include in the model corresponds to no extinction, which is the Yule (pure birth) equivalent:

```
eps.anc = c(0,0,0,0)
```

Fairly straightforward, but why do we do it this way? This particular format makes it easier to generate a large set of nested models to test. We now include an internal function `GetAllModels.anc()` that provides all possible combinations of free parameters nested within the full hisse model.

## Setting up the transition rate matrix

The transition rate matrix is set up separate from the diversification rate parameters. This was intentional. We want to reinforce the idea that SSE models are not trait evolution models, but rather joint models for the tree and the evolution of a trait. It is for this reason that we depict the transition rates in the canonical **Q** matrix format.

To generate the index matrix describing the free parameters in the transition model, we use the `TransMatMaker()` function:

```
trans.rates = TransMatMaker.old(hidden.states=TRUE)
trans.rates
```

```
##      (0A) (1A) (0B) (1B)
## (0A)   NA    4    7   10
## (1A)    1   NA    8   11
```

```
## (OB)    2    5   NA   12
## (1B)    3    6    9   NA
```

Note that the default matrix contains 12 free parameters, which includes dual transitions between both the observed trait and the hidden trait (e.g.,  $q_{0A} \leftrightarrow q_{1B}$ ). Personally, I'm skeptical about such transitions and always remove them from the model entirely. I accomplish this by using the internal function `ParDrop()`:

```
trans.rates.nodual = ParDrop(trans.rates, c(3,5,8,10))
trans.rates.nodual
```

```
##      (OA) (1A) (OB) (1B)
## (OA)   NA    3    5    0
## (1A)    1   NA    0    7
## (OB)    2    0   NA    8
## (1B)    0    4    6   NA
```

You may also want to run a model where we assume all transitions are equal to one another. This will often be a sensible approach because it is generally difficult to reasonably estimate the transition rates (see Beaulieu and O'Meara 2016). To set the rates equal we can use another internal function, `ParEqual()`. This function takes pairs of indexes and sets the two equal based on the index of the first entry. Take our example above and let's set parameter 1 and 6 to have the same rate:

```
trans.rates.nodual.equal16 = ParEqual(trans.rates.nodual, c(1,6))
trans.rates.nodual.equal16
```

```
##      (OA) (1A) (OB) (1B)
## (OA)   NA    3    5    0
## (1A)    1   NA    0    6
## (OB)    2    0   NA    7
## (1B)    0    4    1   NA
```

Note that the index for parameter 6 has become 1 – in other words, their rate will take the same value. Now, let's set all rates to be equal using this function:

```
trans.rates.nodual.allegal = ParEqual(trans.rates.nodual, c(1,2,1,3,1,4,1,5,1,6,1,7,1,8))
trans.rates.nodual.allegal
```

```
##      (OA) (1A) (OB) (1B)
## (OA)   NA    1    1    0
## (1A)    1   NA    0    1
## (OB)    1    0   NA    1
## (1B)    0    1    1   NA
```

Here we've set the index 2 to equal 1, 3 to equal 1, 4 to equal 1, etc. Of course, if this function is confusing, there are other ways to do the same thing. For example, one could do:

```
trans.rates.nodual.allegal = trans.rates.nodual
trans.rates.nodual.allegal[!is.na(trans.rates.nodual.allegal) &
                           !trans.rates.nodual.allegal == 0] = 1
trans.rates.nodual.allegal
```

```
##      (OA) (1A) (OB) (1B)
## (OA)   NA    1    1    0
## (1A)    1   NA    0    1
## (OB)    1    0   NA    1
## (1B)    0    1    1   NA
```

Also note that in order to run a BiSSE model in HiSSE, the matrix set up would look like this:

```
trans.rates.bisse = TransMatMaker.old(hidden.states=FALSE)
trans.rates.bisse
```

```
##      (0) (1)
## (0)  NA  2
## (1)   1 NA
```

Whatever transition matrix is designed, it is supplied to the `trans.rate=` argument in the `hisse()` call:

```
pp = hisse.old(phy, sim.dat, f=c(1,1), hidden.states=TRUE, turnover.anc=turnover.anc,
               eps.anc=eps.anc, trans.rate=trans.rates.nodual.allequal, sann=FALSE)
```

## A common mistake

I wanted to highlight a common mistake I've seen through my email correspondence with several users. It may be of interest to test a model where the hidden state is associated with only a single observed state, such that the model contains states 0A, 1A, and 1B. The diversification parameters might look something like this:

```
turnover.anc = c(1,2,0,3)
eps.anc = c(1,2,0,3)
```

The 0 in the 3rd entry for state 0B designates that the parameter is removed entirely from the model. A common mistake is that the transitions to and from 0B are not removed from the transition matrix. This needs to be done manually:

```
trans.rates <- TransMatMaker.old(hidden.states=TRUE)
trans.rates.nodual.no0B <- ParDrop(trans.rates, c(2,3,5,7,8,9,10,12))
trans.rates.nodual.no0B
```

```
##      (0A) (1A) (0B) (1B)
## (0A)   NA   2   0   0
## (1A)    1  NA   0   4
## (0B)    0   0  NA   0
## (1B)    0   3   0  NA
```

## Changing the output

By default, `hisse` outputs turnover and extinction fraction. I recognize, of course, that this defies the convention of seeking to estimate and interpretation differences in net diversification ( $r_i = \lambda_i - \mu_i$ ). Therefore, users can alter how the final parameters are printed to the screen using the `output.type=` argument in the call `hisse()`. We've included three options: "turnover", "net.div", and "raw", which outputs the results as estimates of speciation ( $\lambda_i$ ) and extinction ( $\mu_i$ ). Thus, to output net diversification:

```
pp = hisse.old(phy, sim.dat, f=c(1,1), hidden.states=TRUE, turnover.anc=turnover.anc,
               eps.anc=eps.anc, trans.rate=trans.rates.nodual.allequal, output.type="net.div", sann=FALSE)
```

## Setting up the 2-state character-independent (CID-2) model

In Beaulieu and O'Meara (2016), we proposed two character-independent models as a means of providing a fairer comparison between trait-dependent and trait-independent models of diversification (for more details see the *Type I errors, Model rejection, & HiSSE vs. FiSSE* vignette). These character-independent (CID) models explicitly assume that the evolution of a binary character is independent of the diversification process without forcing the diversification process to be constant across the entire tree. The first model, which we refer to as "CID-2", contains four diversification process parameters that account for trait-dependent

diversification solely on the two states of an unobserved, hidden trait. In this way, CID-2 contains the same amount of complexity in terms of diversification as a BiSSE model. The second model, which we refer to as “CID-4” contains the same number of diversification parameters as in the general HiSSE model that are linked across four hidden states. In the case of the CID=4 model, we have implemented a separate function, `hisse.null4()` that should be fairly self-explanatory. However, rather than implementing a separate function for the “CID-2” model, we found it is easier to just set it up and test it using the `hisse()` function. This section describes how to do this.

Remember, the goal is to set up a model where the diversification process is independent from the observed states (0 or 1) of the focal trait. In other words, diversification rate differences, if they exist, will only be associated with one of the hidden states (A or B) regardless of the state of the focal trait. Thus, the free parameters for diversification would look like this:

```
turnover.anc = c(1,1,2,2)
eps.anc = c(1,1,2,2)
```

In other words, we are specifying that both 0A and 1A have one set of diversification rates, and 0B and 1B have another. That’s it. This is the “null-two” model.

In regards to the transition rates, there are various ways in which they can be set up. We could also assume all rates are equal:

```
trans.rates = TransMatMaker.old(hidden.states=TRUE)
trans.rates.nodual = ParDrop(trans.rates, c(3,5,8,10))
trans.rates.nodual.allequal = ParEqual(trans.rates.nodual, c(1,2,1,3,1,4,1,5,1,6,1,7,1,8))
trans.rates.nodual.allequal
```

```
##      (OA) (1A) (OB) (1B)
## (OA)  NA   1   1   0
## (1A)   1  NA   0   1
## (OB)   1   0  NA   1
## (1B)   0   1   1  NA
```

We could also specify three rates: one rate describing transitions among the different hidden states (A<->B, which could be interpreted as the rate by which shifts in diversification occur), and two rates for transitions between the observed character states (0->1 or 1->0). Unfortunately, this requires a little bit of clunky coding:

```
# Now we want three specific rates:
trans.rates.nodual.threerates <- trans.rates.nodual
# Set all transitions from 0->1 to be governed by a single rate:
to.change <- cbind(c(1,3), c(2,4))
trans.rates.nodual.threerates[to.change] = 1
# Now set all transitions from 1->0 to be governed by a single rate:
to.change <- cbind(c(2,4), c(1,3))
trans.rates.nodual.threerates[to.change] = 2
# Finally, set all transitions between the hidden state to be a single rate (essentially giving
# you an estimate of the rate by which shifts in diversification occur:
to.change <- cbind(c(1,3,2,4), c(3,1,4,2))
trans.rates.nodual.threerates[to.change] = 3
trans.rates.nodual.threerates
```

```
##      (OA) (1A) (OB) (1B)
## (OA)  NA   1   3   0
## (1A)   2  NA   0   3
## (OB)   3   0  NA   1
## (1B)   0   3   2  NA
```

As before, `turnover.anc`, `eps.anc`, and the transition rate matrix are supplied as arguments to `hisse()`:

```
pp = hisse.old(phy, sim.dat, f=c(1,1), hidden.states=TRUE, turnover.anc=turnover.anc,
               eps.anc=eps.anc, trans.rate=trans.rates.nodual.allequal, sann=FALSE)
```

and the results can be compared against any model in the `hisse` set, including `BiSSE`.

## Plotting `hisse` reconstructions

Our `HiSSE` package provides plotting functionality in `plot.hisse.states()` for character state reconstructions of `class hisse.states` output by our `MarginRecon()` function. Specifically, the function provides an overlay of the state reconstructions on the rate estimates. There are a couple of options for how these reconstructions to be plotted. First, a single `hisse.states` object can be supplied and it will provide a heat map of the diversification rate parameter of choice. Users can choose among turnover, net diversification (“net.div”), speciation, extinction, or extinction fraction (“extinction.fraction”). I’ve provided example `hisse.states` output from the example data set simulated above. This particular model assumed two diversification rate parameters – i.e., `turnover.anc=c(1,1,1,2)`, and `eps.anc=c(1,1,1,1)`. Let’s load this file and check that everything has loaded correctly and is of the proper `hisse.states` class:

```
# pp.recon <- MarginRecon(phy=phy, data=sim.dat, f = pp$f, pars = pp$solution
#                           , hidden.states = pp$hidden.states, aic = pp$AIC)
load("testrecon1.Rsave") # Line above shows the command to create this result.
class(pp.recon)
```

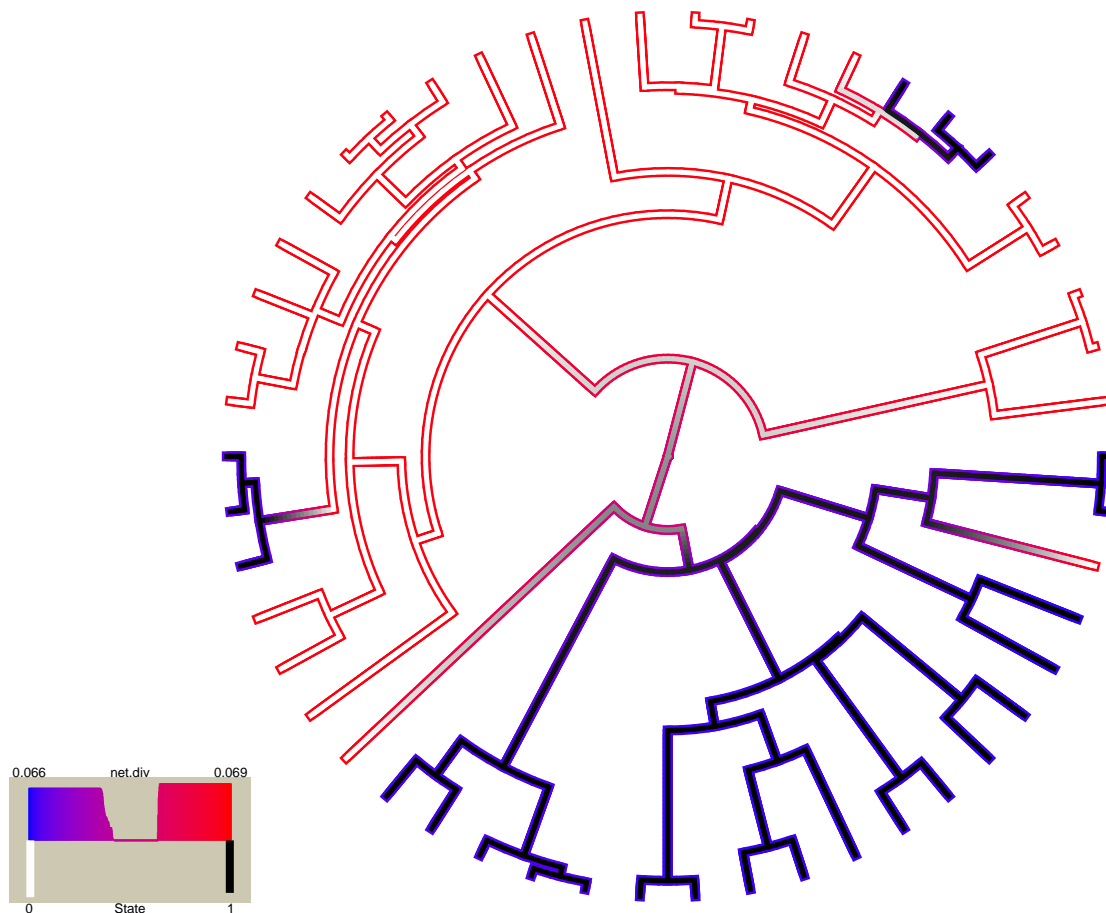
```
## [1] "hisse.states"
```

```
pp.recon
```

```
##
## Phylogenetic tree with 50 tips and 49 internal nodes.
##
## Tip labels:
##  sp15, sp57, sp58, sp49, sp50, sp42, ...
## Node labels:
##  1, 3, 1, 1, 1, 1, ...
##
## Rooted; includes branch lengths.
```

Now that we have the right files and information, let’s plot net diversification rates:

```
plot.hisse.states(pp.recon, rate.param="net.div", show.tip.label=FALSE)
```



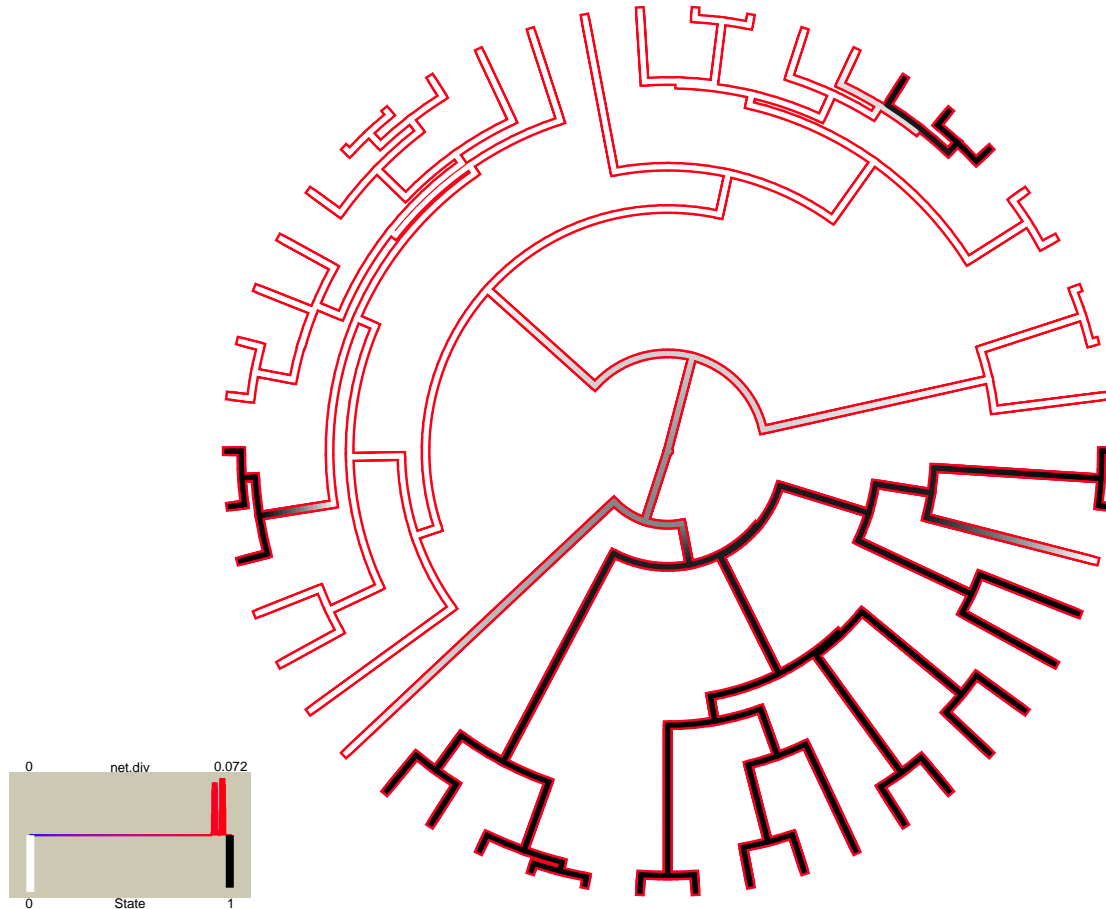
```
## $rate.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0.066161, 0.069176).
##
##
## $state.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0, 1.001).
```

These are the default colors: red to blue for rate and white to black for state. However, other colors can be specified (see `plot.hisse.states` manual). The legend in the bottom left corner provides the frequency of the observed states (0 or 1) and the distribution of net diversification rates at the tips only. Note that in this particular model I only specified two diversification rate parameters [i.e., `turnover.anc=c(1,1,1,2)`, and `eps.anc=c(1,1,1,1)`], and yet there seems to be a continuous range of rates at the tips. What gives? Well, when a single reconstruction is provided the rates painted on each branch are the weighted average of the rate, with the marginal probability used as the weights. So this particular painting has taken into account the uncertainty in rates on each branch.

But, please be careful here! Notice that there are parts of the tree that have much higher rates (denoted by bright red) than others (denoted by darker blue). This is actually highly misleading. If you look at the legend

the rates span from 0.066 - 0.069! So, really, there aren't any meaningful differences in the diversification rates despite what the painting may say. By default the visualization uses the minimum rate on the tree for the minimum color, and the maximum rate for the maximum color. However, users may want to use the same color scale across models, even if some of them have a smaller range than others. A vector with the minimum and maximum rate across all models can be passed to the visualization:

```
plot.hisse.states(pp.recon, rate.param="net.div", show.tip.label=FALSE,
                 rate.range=c(0,0.072))
```



```
## $rate.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0, 0.072072).
##
##
## $state.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0, 1.001).
```

Now the differences completely disappear.

A really cool feature of the plotting function is that if you provide a list of `hisse.states` objects the function



will “model-average” the results. In other words, branches are painted such that they take into account both state and rate uncertainty and uncertainty in the model. The `plot.hisse.states()` first calculates a weighted average of the likeliest state and rate combination for every node and tip for each model in the set, using the marginal probability as the weights, which are then averaged across all models using the Akaike weights (i.e., the relative probability of a model).

A first initial step when doing the modeling-averaging approach is to make sure that the `hisse.states` objects contain the AIC from the model fit embedded in it. The plotting function will not work without it:

```
pp.recon$aic
```

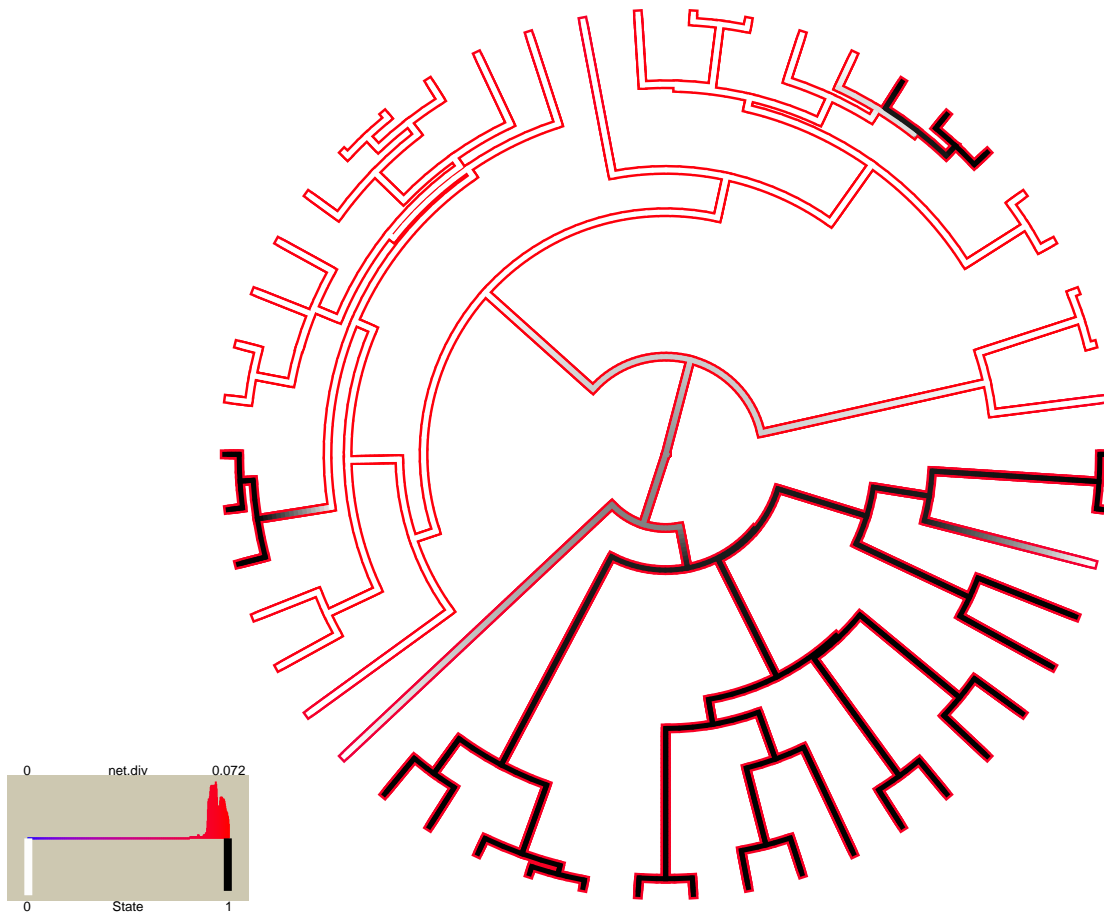
```
## [1] 364.8615
```

If this returns a NULL, then something has gone wrong and you should check how you performed the reconstructions. The AIC for the model can be supplied as an argument in the `MarginRecon()` function (using the `pp` object we defined above):

```
pp.recon = MarginRecon(phy, sim.dat, f=c(1,1), hidden.states=TRUE, pars=pp$solution,
                      aic=pp$aic, n.cores=2)
```

I’ve created two additional `hisse.states` objects that I will use to demonstrate how to plot model-averaged states and rates: one reconstruction is based on the null-two model [`turnover.anc=c(1,1,2,2)`], and the other assumes four free turnover rates [i.e., `turnover.anc=c(1,2,3,4)`]; in all cases I assumed equal transition rates and equal extinction fractions. I recognize that there are many ways to generate a list. But here is one way, where I’m assuming that the marginal reconstructions from three models are saved to the directory we are working from:

```
hisse.results.list = list()
load("testrecon1.Rsave")
hisse.results.list[[1]] = pp.recon
load("testrecon2.Rsave")
hisse.results.list[[2]] = pp.recon
load("testrecon3.Rsave")
hisse.results.list[[3]] = pp.recon
# Now supply the list the plotting function
plot.hisse.states(hisse.results.list, rate.param="net.div", show.tip.label=FALSE, rate.range=c(0,0.072))
```



```
## $rate.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0, 0.072072).
##
##
## $state.tree
## Object of class "contMap" containing:
##
## (1) A phylogenetic tree with 50 tips and 49 internal nodes.
##
## (2) A mapped continuous trait on the range (0, 1.001).
```

Although this is similar to the plot above, which shows no real rate differences, it actually accounts for both the uncertainty in the model as well as the reconstructions. Note that there are many features that can be adjusted in `plot.hisse.states()`, which is described in detail in the manual.

Note that the above code is obviously a pretty lame way of generating a list. An even easier way would be to do something like this:

```
# First, suck in all the files with .Rsave line ending in your working directory:
# files = system("ls -1 | grep .Rsave", intern=TRUE) # This will not run in a Windows system.
files = dir( pattern = "*.Rsave" ) # This is system independent.
# Create an empty list object
```

```
hisse.results.list = list()
# Now loop through all files, adding the embedded pp.recon object in each
for(i in sequence(length(files))){
  load(files[i])
  hisse.results.list[[i]] = pp.recon
}
rm(pp.recon)
```

## References

- Beaulieu, J.M., and B.C. O'Meara. (2016). Detecting hidden diversification shifts in models of trait-dependent speciation and extinction. *Syst. Biol.* 65:583-601.
- FitzJohn R.G. 2012. Diversitree: comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution* 3:1084-1092.