A newer, faster HiSSE function

Jeremy M. Beaulieu

Getting started

As of version 1.9.6, we now provide a new set of functions that execute a faster version of the HiSSE model described by Beaulieu and O'Meara (2016). Here we implement a more efficient means of carrying out branch calculations, so it is exceptionally faster than in previous versions. Specifically, we break up the tree into sets of branches whose branch calculations are independent of one another. We then carry out all descendent branch calculations simultaneously, combine the probabilities based on their shared ancestry, then repeat for the next set of descendent branches. In testing, we've found that as the number of taxa increases, the calculation becomes much more efficient. Another main differences here is that the new `hisse()` function allows up to four hidden categories, which means that both the character-independent model (`hisse.null4()`) is no longer a standalone function and can be set up directly in the same way as we can setup a BiSSE, HiSSE, or CID-2 models. It is our hope that this will emphasize to users the direct connection between BiSSE, HiSSE, and our CID models.

Below I will demonstrate how to set up various models, and how the likelihoods compare against those from `diversitree` whenever possible. Before getting started, be sure to load the `hisse` and `diversitree` packages:

```
## Loading required package: ape
```

```
## Loading required package: deSolve
```

```
## Loading required package: GenSA
```

```
## Loading required package: subplex
```

```
## Loading required package: nloptr
```

Simulating a practice data set

I will first 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
```

Setting up a BiSSE model using HiSSE

As with the original HiSSE implementation (see `hisse.old()`), the number of free parameters in the model for both 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 always ordered as follows for a given hidden state, i: 0_i, 1_i. However, in this case we do not want any hidden states. But first let's set up the "dull null" – i.e., turnover and extinction fraction are the same for both states. Note the "f" represents the sampling fraction for each observed state combination, which is a vector ordered in the same manner as for turnover and extinction fraction vectors:

```
turnover <- c(1,1)
extinction.fraction <- c(1,1)
f <- c(1,1,1,1)
```

Next, we have to set up a transition matrix. There is a function provided to make this easy, and allows users to customize the matrix to fit particular hypotheses. Be sure to look at the options on this function call, for allowing diagonals and for customizing the matrix when you have character independent model.

```
trans.rates.bisse <-  TransMatMakerHiSSE(hidden.traits=0)
print(trans.rates.bisse)
```

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

Now, we can call HiSSE and estimate the parameters under this model using the default settings:

```
dull.null <- hisse(phy=phy, data=sim.dat, f=f, turnover=turnover,
                   eps=extinction.fraction, hidden.states=FALSE,
                   trans.rate=trans.rates.bisse, sann=FALSE)
```

If you wanted to set up a true BiSSE model, where the turnover rate parameters are unlinked across the observed state combinations, you would simply do the following:

```
turnover <- c(1,2)
extinction.fraction <- c(1,1)
BiSSE <- hiSSE(phy=phy, data=sim.dat, f=f, turnover=turnover,
                   eps=extinction.fraction, hidden.states=FALSE,
                   trans.rate=trans.rates.bisse)
```

Setting up a HiSSE model

Setting up a character-dependent HiSSE model is relatively straightforward, and relies on all the same tools as above. One important thing to bear in mind, is that again, the states are ordered by state combination within each hidden state. For example, if you want two hidden states, A and B, the order of the parameters in the model is 0A, 1A, 0B, and 1B. So, in this case, we just need to specify the free parameters we want for diversification. Here we are just going assume turnover varies across the different states:

```
turnover <- c(1,2,3,4)
extinction.fraction <- rep(1, 4)
f = c(1,1)
```

We also have to extend the transition rate matrix. This is done by specifying the number of hidden states:

```
trans.rate.hisse <- TransMatMakerHiSSE(hidden.traits=1)
print(trans.rate.hisse)
```

```
##      (0A) (1A) (0B) (1B)
```

```
## (0A)   NA    2    5   NA
## (1A)    1   NA   NA    5
## (0B)    5   NA   NA    4
## (1B)   NA    5    3   NA
```

Now, we can just plug these options into MuHiSSE:

```
HiSSE <- hisse(phy=phy, data=states.trans, f=f, turnover=turnover,
                    eps=extinction.fraction, hidden.states=TRUE,
                    trans.rate=trans.rate.hisse)
```

Setting up a character-independent HiSSE model (i.e., CID-2, CID-4)

Remember, for any character-independent model, the diversification rates *must* be decoupled from the observed states. So, to do this, we simply set the diversification rates to be equal for all states for a given hidden state. Below, I will show how to do this for a character-independent model with two rate shifts in the tree, what we refer to as the CID-2 model:

```
turnover <- c(1, 1, 2, 2)
extinction.fraction <- rep(1, 4)
f = c(1,1)
trans.rate <- TransMatMakerHiSSE(hidden.traits=1, make.null=TRUE)
```

For the transition rate matrix, I included a setting called `make.null` that simply replicates the transition model across the hidden states. This way the transition rates are not impacted by changes in the diversification rate regime – that is, $q_{0,A \rightarrow 1,A} = q_{1,B \rightarrow 0,B}$.

Here is a how you would set up the diversification rates for models with three hidden state, which is equivalent to CID-4 model:

```
turnover <- c(1, 1, 2, 2, 3, 3, 4, 4)
extinction.fraction <- rep(1, 8)
trans.rate <- TransMatMakerHiSSE(hidden.traits=3, make.null=TRUE)
```

Some considerations regarding optimization

We've done our best to ensure that the results obtain from `hisse()` and other functions (`MuHiSSE()`, `GeoHiSSE()`, and `MiSSE()`) are reliable and at their MLE *most* of the time. Like any likehood-based comparative method, there will be times where the optimizer might terminate early, when the starting values put the optimizer on a track to get stuck on a suboptimal local peak. There will even be times will be times when the branch calculations will fail and return nonsensical results. We implemented checks internally throughout to catch some of these issue when they arise. However, to ensure a robust parameter search we've now made the two-step (`sann=TRUE`) optimization routine as the default for all functions within hisse. This first step involves a simulated annealing (SA) optimization routine. An SA algorithm is basically hill-climbing that is ideal for navigating likelihood surfaces that have many local peaks to find the global peak. The algorithm works by not always picking the best move, but rather a random move. If the selected move improves the solution, then it is always accepted. If not, the algorithm makes the move anyway with some probability less than 1. The probability decreases exponentially with how bad a move is scaled by the "temperature" of the chain. At higher temperatures bad moves are more likely, and at lower temperatures bad moves are less likely. The second step of our routine refines the SA search using the standard subplex routine that used to be the default. That is, the starting values in step 2 are the ML parameter estimates from the SA search. Users can, of course, revert back to the original "fast and loose" optimization routine by simply doing `sann=FALSE`.

There are other considerations that users should make when evaluating the robustness of their results. For instance, if users choose `sann=FALSE`, does the choice of starting values impact the results? If so, then it might be worth trying a large set of starting values to find the set that produces the best likelihood. Also, to increase speed, it might be wise to reduce the size of the bounds. Right now the bounds on turnover (`turnover.upper`) and transition rates (`trans.upper`) are set pretty high. Shrinking the bounds will limit the optimizer from spending time in really bad areas of parameter space.

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.