Using BiSSE with diversitree

Rich G. FitzJohn

May 7, 2009

1 Introduction

This demo is in a very early stage of development. It does not aim to be a complete reference to the package, but should give some useful hints. The code all assumes that the diversitree package is loaded.

```
> library(diversitree)
```

2 Simulating phylogenies

diversitree includes code for simulating phylogenies, under both a character-dependent BiSSE-type model and a character-independent model (as in Nee et al., 1994). The tree simulation code is likely to change a reasonable amount soon. It will be useful to have a simulated tree to demonstrate the code; the following generates a birth-death tree following the BiSSE model with 203 species, using $\lambda_0 = 0.1$, $\lambda_1 = 0.2$, $\mu_0 = \mu_1 = 0.03$, and $q_{01} = q_{10} = 0.01$, starting in state 0 and running for 60 time units.

```
> pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
> set.seed(2)
> phy <- tree.bisse(pars, max.t = 60, x0 = 0)
```

3 Running BiSSE

The way diversitree runs BiSSE, you first construct a likelihood function with `make.bisse`, then use this in a maximum likelihood or MCMC approach to do your inference. The `make.bisse` takes as its first two arguments a tree and set of character states:

```
> lik <- make.bisse(phy, phy$tip.state)
```

The object "`lik`" is the likelihood function; given a vector of parameters (in the order λ_0, λ_1, μ_0, μ_1, q_{01}, q_{10}) it computes the likelihood of the tree and character data:

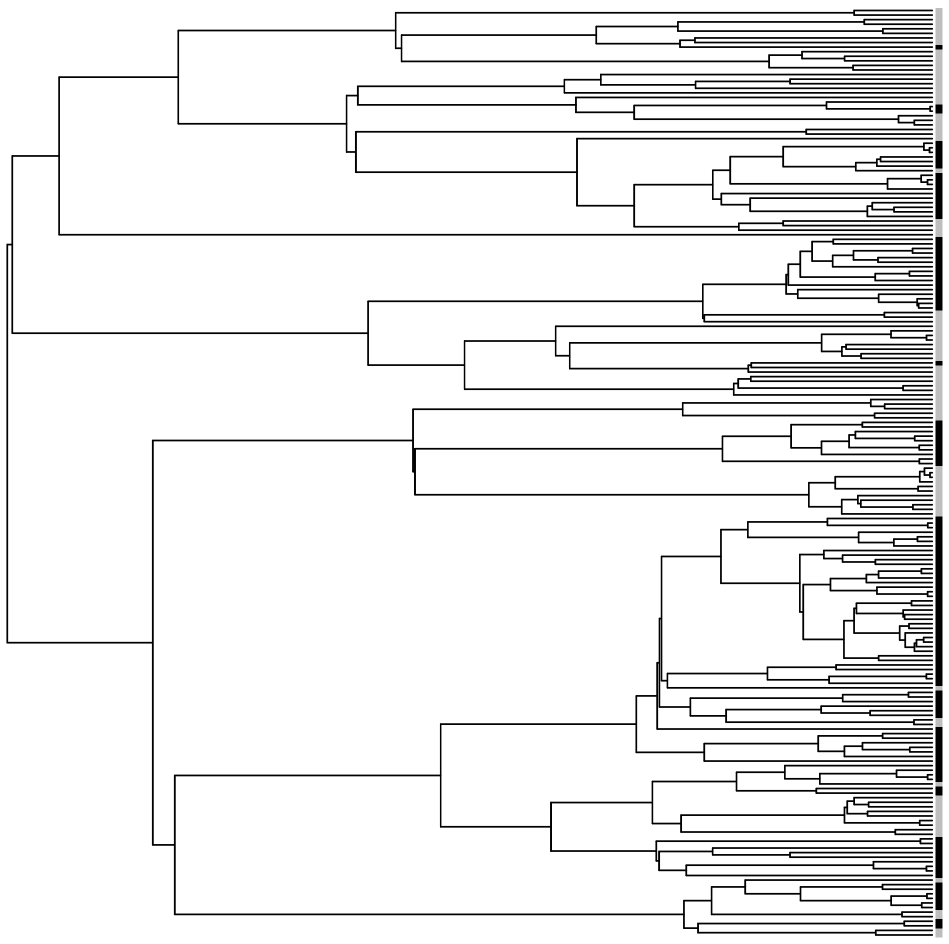

Figure 1: A random BiSSE tree. Gray indicates state `0` (low speciation) and black indicates state 1 (high speciation).

```
> lik(pars)

[1] -660.0173
```

To do an maximum likelihood analysis, we need to find the set of parameters that maximises the likelihood. The `find.mle` function provides an interface to R's built-in `optim` that simplifies this process. To perform an optimisation, we need a starting point. The `starting.point` function does a very poor heuristic search for a sensible starting point, based on the character-independent birth-death fit.

```
> p <- starting.point(phy)
> p

   lambda0    lambda1        mu0        mu1        q01        q10
0.16721949 0.16721949 0.08017315 0.08017315 0.01740927 0.01740927
```

Run the optimisation (this may take up to a minute, depending on the speed of your computer).

```
> fit <- find.mle(lik, p)
> fit

$par
    lambda0     lambda1         mu0         mu1         q01         q10
0.099400026 0.196255619 0.023972305 0.032610865 0.010040429 0.009863226

$lnLik
[1] -659.9226

$counts
function gradient
      54       54

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"

attr(,"class")
[1] "mle"       "mle.bisse"
```

The object returned by `find.mle` will change in the future, but the key bits are the element `par` with the ML parameters and `lnLik` with the best log-likelihood value. It would be preferable to start this search from multiple starting points, and I will add something to automate that to some degree soon.

3.1 Constrained models

There are two ways of creating constrained models; `constrain.par` constrains some parameters to be equal to other parameters, and `fix.par` assigns parameters to be equal to a particular value. These are discussed to some degree on their help pages. As an example, consider the model where $\lambda_1 = \lambda_0$:

```
> lik.equal.l <- constrain.par(lik, c(NA, 1, NA, NA, NA, NA))
```

The second argument to `constrain.par` describes the constraint; values that are `NA` are "free" (will be optimised), and parameters with integer values "point" at other parameters (in this case, the second parameter, λ_1 points at λ_0). This can be optimised as before, starting this time at the full model's MLE:

```
> fit.equal.l <- find.mle(lik.equal.l, fit$par)
> fit.equal.l[1:2]

$par
    lambda0         mu0         mu1         q01         q10
0.162577038 0.107539146 0.000000000 0.006106392 0.013016537

$lnLik
[1] -663.4042
```

(the `lik.equal.l` function really only takes a five-element parameter vector, but `find.mle` filters this to remove the second element automatically). This model has tweaked the extinction rates to get a suitable diversification rate, and settled on a λ that is intermediate between the true rates. The likelihood has dropped 3.5 units; a likelihood ratio test gives this a p value of 0.008:

```
> 1 - pchisq((fit$lnLik - fit.equal.l$lnLik) * 2, 1)

[1] 0.00832035
```

In a second example, consider preventing $0 \rightarrow 1$ transitions:

```
> lik.no.01 <- fix.par(lik, c(NA, NA, NA, NA, 0, NA))
> fit.no.01 <- find.mle(lik.no.01, fit$par)
> fit.no.01[1:2]

$par
   lambda0    lambda1        mu0        mu1        q10
0.10568830 0.23545089 0.05834892 0.14417997 0.02559433

$lnLik
[1] -673.0052
```

This model is strongly rejected ($p < 0.001$)

```
> 1 - pchisq((fit$lnLik - fit.no.01$lnLik) * 2, 1)

[1] 3.134327e-07
```

3.2 Markov Chain Monte Carlo (MCMC)

The likelihood function can also be used to perform Markov Chain Monte Carlo (MCMC). I will use an exponential prior with a rate of 5.7 (having a mean twice the character independent diversification rate). Running this will take some time (allow up to 1 s per step on this tree). The output is a large data frame with the contents of the chain.

```
> r <- 1/(2 * (p[1] - p[3]))
> ans <- mcmc(lik, fit$par, nsteps = 100, w = rep(0.1, 6), lower = rep(0,
+     6), upper = rep(Inf, 6), prior = r, fail.value = -Inf)
```

4 Skeleton trees

To sample from the simulated phylogeny above to make a skeleton tree, we can do this:

```
> set.seed(1)
> n.taxa <- length(phy$tip.label)
> keep <- sort(sample(n.taxa, 50))
> phy.s <- drop.tip(phy, setdiff(seq_len(n.taxa), keep))
```

A need likelihood function is needed for this tree. We also need to specify what the sampling fraction is (close to 20% for both).

```
> n.real <- table(phy$tip.state)
> n.samp <- table(phy.s$tip.state[phy.s$tip.label])
> sampling.f <- n.samp/n.real
> sampling.f

        0         1
0.2048193 0.2750000
```

Pass this in to `make.bisse` and construct a new likelihood function that accounts for the sampling:

```
> lik.s <- make.bisse(phy.s, phy.s$tip.state, sampling.f = sampling.f)
```

This can then be optimised, as before:

```
> fit.s <- find.mle(lik.s, p)
> fit.s[1:2]

$par
    lambda0     lambda1         mu0         mu1         q01         q10
0.080669502 0.227324661 0.000000000 0.080338704 0.010907117 0.005236501

$lnLik
[1] -193.9340
```

The fits compare reasonably well from the full tree and sampled tree

```
> round(rbind(full = fit$par, sampled = fit.s$par), 3)
```

```
        lambda0 lambda1   mu0   mu1   q01   q10
full      0.099   0.196 0.024 0.033 0.010 0.010
sampled   0.081   0.227 0.000 0.080 0.011 0.005
```

5 Terminally unresolved trees

For this section, I will show how the shorebird example in the manuscript can be performed. The file `Thomas-tree.nex` contains the shorebird supertree assembled by Thomas et al. (2004) in nexus format and the file `Lislevand-states.csv` contains a measure of the level of sexual dimorphism, computed from the data in Lislevand et al. (2007). The tree contains many polytomies, so I am converting it into a terminally unresolved tree, with the `polytomies.to.clades` function.

```
> tree <- read.nexus("data/Thomas-tree.nex")
> tree2 <- polytomies.to.clades(tree)
```

Terminally unresolved trees can be assembled manually from a phylogeny that contains only the exemplar tips using the `make.clade.tree` function.

```
> states <- read.csv("data/Lislevand-states.csv", as.is = TRUE)
> states <- structure(states$dimorph, names = states$species)
```

Every species needs tip data

```
> states <- states[tree$tip.label]
> names(states) <- tree$tip.label
```

This is all the information `make.bisse` needs. Construct a likelihood function by categorising states with 15% dimorphism being the cut-off:

```
> lik.sb <- make.bisse(tree2, abs(states) > 0.15)
```

This starting point comes from ape's function `bd.ext`, but this cannot be automated, as it does not return an object.

```
> p <- c(0.147, 0.147, 0, 0, 0.02, 0.02)
> names(p) <- c("lambda0", "lambda1", "mu0", "mu1", "q01", "q10")
```

Find the ML point (this will take 1 − −2 minutes, and will produce warnings that can be ignored).

```
> fit.sb <- find.mle(lik.sb, p)
> fit.sb[1:2]
```

```
$par
   lambda0    lambda1        mu0        mu1        q01        q10
0.05303850 0.20341423 0.00000000 0.00000000 0.04235417 0.28197098

$lnLik
[1] -633.7833
```

It looks from this that the speciation rate and the character transition rate in state 1 are higher than the corresponding rates in state 0 ($\lambda_1 > \lambda_0$, $q_{10} > q_{01}$). The significance of this can be tested with a likelihood ratio test:

```
> lik.sb.l <- constrain.par(lik.sb, c(NA, 1, NA, NA, NA, NA))
> lik.sb.q <- constrain.par(lik.sb, c(NA, NA, NA, NA, NA, 5))
> fit.sb.l <- find.mle(lik.sb.l, fit.sb$par)
> fit.sb.q <- find.mle(lik.sb.q, fit.sb$par)
```

These constrained models are significantly worse fits than the full model (λ: $p = 0.0005$, q: $p = 0.01$).

```
> 1 - pchisq((fit.sb$lnLik - fit.sb.l$lnLik) * 2, 1)

[1] 0.0004801241

> 1 - pchisq((fit.sb$lnLik - fit.sb.q$lnLik) * 2, 1)

[1] 0.01276910
```

This analysis can also be done with MCMC (not shown)

```
> ans <- mcmc(lik.sb, fit.sb$par, nsteps = 10000, w = rep(0.1,
+     6), lower = rep(0, 6), upper = rep(Inf, 6), prior = 1/p[1] *
+     2, fail.value = -Inf)
```

References

Lislevand, T., J. Figuerola, and T. Székely. 2007. Avian body sizes in relation to fecundity, mating system, display behavior, and resource sharing. Ecology 88:1605.

Nee, S., R. M. May, and P. H. Harvey. 1994. The reconstructed evolutionary process. Philos. Trans. R. Soc. Lond. B Biol. Sci. 344:305–311.

Thomas, G. H., M. A. Wills, and T. Székely. 2004. A supertree approach to shorebird phylogeny. BMC Evolutionary Biology 4:28.