Using corHMM 2.0

James D. Boyko and Jeremy M. Beaulieu

Introduction

The original version of the R-package `corHMM` contained a number of distinct functions for conducting analyses of discrete morphological characters. This included the `corHMM()` function for fitting a hidden rates model, which uses "hidden" states as a means of allowing transition rates in a binary character to vary across a tree. In reality, the hidden rates model falls within a general class of models known as hidden Markov models (HMM), and it need not only be applied to binary characters. The use of hidden states is a way of conceptualizing a rate class. With multiple hidden states, you have multiple rate classes. So, whether the focal trait is binary or contains multiple states, or whether the observed states represents a set of binary and multistate characters, hidden states can be applied as a means of allowing heterogeneity in the transition model. Choosing a model specific to your question is of utmost importance in any comparative method, and in this new version of `corHMM` we provide users with the tools to create their own hidden Markov models.

Before delving into this further it may be worth showing a little of what is underneath the hood, so to speak. To begin, consider a single binary character with states *0* and *1*. If the question was to understand the asymmetry in the transition between these two states, the model, **Q**, would be a simple 2x2 matrix,

$$Q = \begin{bmatrix} - & q_{0-1} \\ q_{1-0} & - \end{bmatrix}$$

This *transition rate matrix* is read as describing the transition rate *from* ROW *to* COLUMN. And as you can see there are just two transitions going from 0 to 1, and from 1 to 0 because there are only two states possible in this model. Now, suppose we have a second character that is also binary. This means that the number of possible states you *could* observe is expanded to account for all the combination of states between two characters – that is, you could observe *00*, *01*, *10*, or *11*. To accommodate this, we need to expand our model now such that it becomes a 4x4 matrix,

$$Q = \begin{bmatrix} - & q_{00-01} & q_{00-10} & q_{00-11} \\ q_{01-00} & - & q_{01-10} & q_{01-11} \\ q_{10-00} & q_{10-01} & - & q_{10-11} \\ q_{11-00} & q_{11-01} & q_{11-10} & - \end{bmatrix}$$

The model is also considerably more complex as the number of transitions in this rate matrix now goes from 2 to 12. However, with these models we often make a simplifying assumption: we do not allow for transitions in two states to occur at the same time. In other words, if a lineage is in state *00* it cannot immediately transition to state *11*, rather, it must first transition either to state *01* or *10* before finally transitioning to state *11*. So, we can simplify the matrix by removing these "dual" transitions from the model completely,

$$Q = \begin{bmatrix} - & q_{00-01} & q_{00-10} & - \\ q_{01-00} & - & - & q_{01-11} \\ q_{10-00} & - & - & q_{10-11} \\ - & q_{11-01} & q_{11-10} & - \end{bmatrix}$$

What we just described is essentially the popular model of Pagel (1994), which tests for correlated evolution between two binary characters. But, one thing that is not obvious: the states in the model need not be represented solely as combinations of binary characters. For example, the focal character may be two characters, like say, flowers that are red with and without petals, and blue flowers with and without petals.

One could just code it as a single multistate character, where *1*=red petals, *2*=red with no petals (i.e., sepals are red), *3*=blue petals, and *4*=blue with no petals (i.e., sepals are blue). The model would then be,

$$Q = \begin{bmatrix} - & q_{1-2} & q_{1-3} & q_{1-4} \\ q_{2-1} & - & q_{2-3} & q_{2-4} \\ q_{3-1} & q_{3-2} & - & q_{3-4} \\ q_{4-1} & q_{4-2} & q_{4-3} & - \end{bmatrix}$$

Notice it is the same as before, but the states are transformed from binary combinations to a multistate character. And to make this point even clearer, we will drop the "dual" transitions,

$$Q = \begin{bmatrix} - & q_{1-2} & q_{1-3} & - \\ q_{2-1} & - & - & q_{2-4} \\ q_{3-1} & - & - & q_{3-4} \\ - & q_{4-2} & q_{4-3} & - \end{bmatrix}$$

Again, exactly the same.

Here we have modified `corHMM()` to transform any character or set of characters into a *single* multistate character. The model can then be expanded to accomodate an arbitrary number of hidden states. Thus, `corHMM()` now contains `rayDISC()` and `corDISC()` capabilities with the added bonus of allow for hidden states. This vignette is comprised of three sections, where we demonstrate all these extensions as well as other new features:

- **Section 1 Default use of corHMM**
  - 1.1: No hidden rate categores
  - 1.2: Any number of hidden rate categories
- **Section 2 How to make and interpret custom models**
  - 2.1: Creating and using custom rate matrices
    - 2.1.1: One rate category
    - 2.1.2: Any number of rate categories
  - 2.2: Some examples of "biologically informed" models
    - 2.2.1: Precursor model
    - 2.2.2: Ordered habitat change
    - 2.2.3: Ontological relationship of multiple characters
  - 2.3: Estimating models when node states are fixed

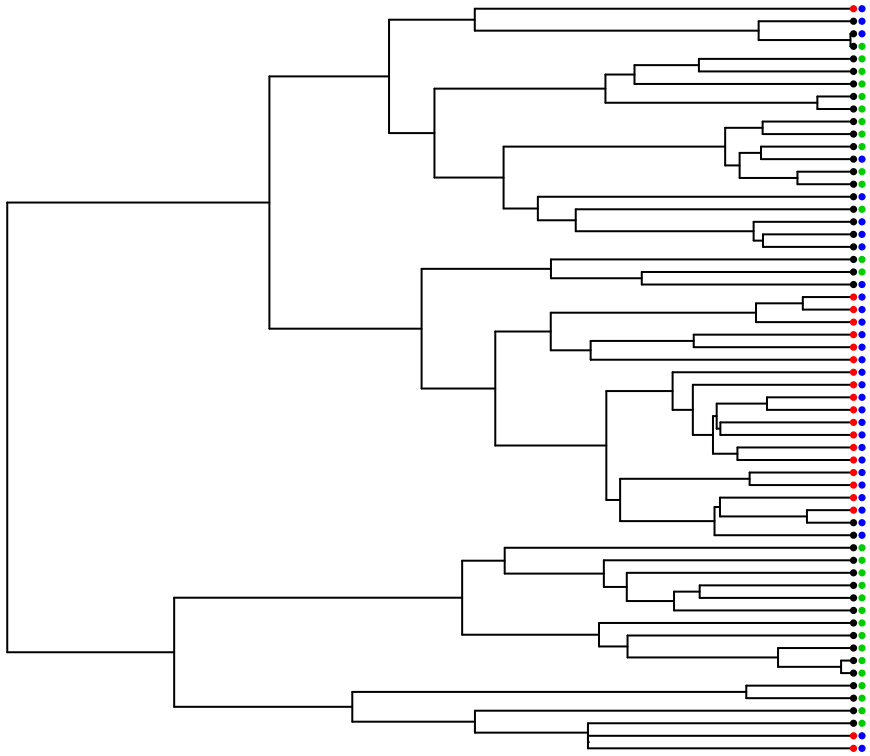
Section 1: Default use of corHMM

1.1: No hidden rate categories

We'll use the primate dataset that comes with `corHMM`, which comes from the empirical example in Pagel and Meade (2006).

```
set.seed(1985)
require(ape)
require(expm)
require(corHMM)
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
```

```
plot(phy, show.tip.label = FALSE)
data.sort <- data.frame(data[, 2], data[, 3], row.names = data[,1])
data.sort <- data.sort[phy$tip.label, ]
tiplabels(pch = 16, col = data.sort[,1]+1, cex = 0.5)
tiplabels(pch = 16, col = data.sort[,2]+3, cex = 0.5, offset = 0.5)
```



We have two characters each with two possible states: Trait 1 is absence (black) or presence (red) of estrus advertisement in females, and trait 2 reflects single male (green) or multimale (blue) mating system in primates.

The default use of `corHMM()` only requires that you declare your *phylogeny*, your *dataset*, and the number of *rate categories* (more detail about this later). We have updated `corHMM()` to handle different types of input data. Previously, the data could only contain two columns: [,1] a column of species names, and [,2] a column of state values. As of `corHMM()`, the first column must be species names (as in the previous version), but there can be any number of data columns. If your dataset does have 2 or more columns of trait information, each column is taken to describe an independently evolving character. Note that when the `corHMM()` call is used, the function automatically determines all the unique character combinations *observed* in the data set. In our primate example only 3 of the 4 possible combinations are observed, and so the model is constructed accordingly. Also, dual transitions are automatically disallowed. In other words, we expect that a species cannot go directly from estrus advertisement being absent in a single male mating system to having estrus advertisement in a multimale mating system. They must first evolve either estrus advertisement or multimale mating system.

Let's give this a try:

```
MK_3state <- corHMM(phy = phy, data = data, rate.cat = 1)
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
```

```
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
MK_3state
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -41.90867 91.81734 92.54462        1   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)
## (1,R1)         NA 0.01898999         NA
## (2,R1) 0.05661172         NA 0.02627317
## (3,R1)         NA 0.01610554         NA
##
## Arrived at a reliable solution
```

When you run your corHMM object you are greeted with a summary of the model. Your model fit is described by the log likelihood (lnL), Akaike information criterion (AIC), and sample size corrected Akaike information criterion (AICc). You are also given the number of rate categories (Rate.cat) and number of taxa (ntax).

The *Rates* section of the output describes transition rates between states and are organized as a matrix. Again, the *transition rate matrix* is read as the transition rate **from** ROW **to** COLUMN. For example, if you were interested in the transition rate from State 1 (i.e., absence of estrus advertisement in a single male mating system) to State 2 (i.e., absence of estrus advertisement in a multimale mating system) you would be looking at the Row 1, Column 2, entry. For a time calibrated ultrametric tree, these rates will depend on the age of your phylogeny.

You should notice that the state legend was printed to the screen when running `corHMM()`. But if you can also obtain the exact coding for each species using the `corHMM()` results object itself. This will provide an augmented dataframe. It takes the initial user data and adds a column that corresponds to how corHMM treated each species:

```
head(MK_3state$data.legend)
```

```
##                  Genus_sp T1 T2 legend
## 1    Cercocebus_torquatus  1  1      3
## 2  Cercopithecus_aethiops  0  1      2
## 3      Cercopithecus_mona  0  0      1
## 4 Cercopithecus_nictitans  0  0      1
## 5      Colobus_angolensis  0  1      2
## 6         Colobus_guereza  0  0      1
```

Alternatively, a user can supply their dataset to getStateMat4Dat which as one of its output provides a legend consistent with the `corHMM()` function. The other output is an index matrix (or rate matrix) which describes which rates are to be estimated in `corHMM` (we'll talk more of the index matrix later):

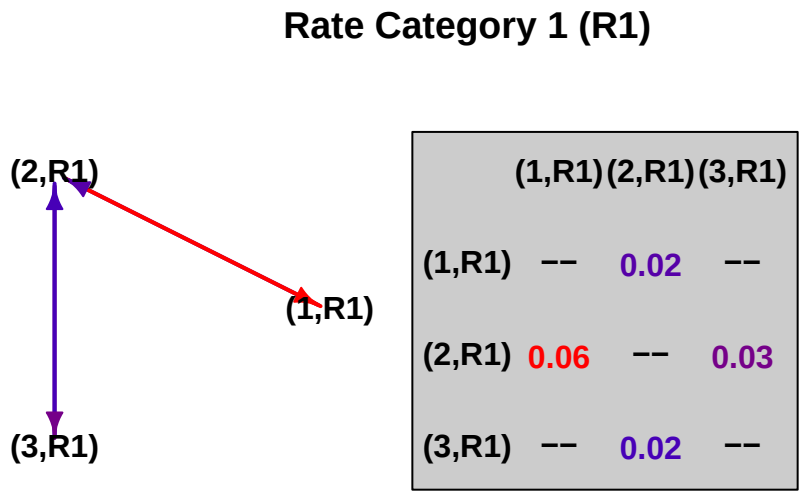
```
getStateMat4Dat(data)
```

```
## $legend
```

```
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   4
## (3)   0   3   0
```

Finally, interpreting a Markov matrix can be difficult, especially when you're just starting out. This problem is compounded when users begin to apply the more complex Hidden Markov models (which is done by setting rate.cat > 1) to their data. To help users we have implemented a new plotting function:

```
plotMKmodel(MK_3state)
```



This function can take a `corHMM` object (which is the result of running `corHMM()`) or a custom rate matrix (discussed in a later section) and plot the model in two parts. On the left is a ball and stick diagram depicting the transitions between the states. On the right is a simplified rate matrix (a rounded version of the solution output of corHMM). The colors of the arrows match the rates.

1.2: A trait with any number of states and any number of hidden rate categories

The major difference between this version of `corHMM` and previous versions is allowing models of any number of states and any number of hidden rate categories (*hidden rate categories will be explained in more depth in section 2*). Running a hidden Markov model (HMM) only requires assigning a value greater than 1 to the rate.cat input. We will use the data from above and assign 2 rate categories.

```
HMM_3state <- corHMM(phy = phy, data = data, rate.cat = 2, model = "SYM")
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
```

```
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
HMM_3state
```

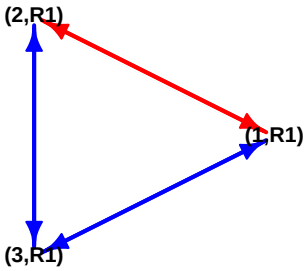
```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -41.30284 98.60568 101.4292        2   60
##
## Rates
##              (1,R1)       (2,R1)      (3,R1)       (1,R2)       (2,R2)
## (1,R1)           NA 1.000000e+02 0.011758075 3.060811e-02           NA
## (2,R1) 100.00000000           NA 0.001535932           NA 3.060811e-02
## (3,R1)   0.01175808 1.535932e-03          NA           NA           NA
## (1,R2)   0.01143993           NA          NA           NA 2.061154e-09
## (2,R2)           NA 1.143993e-02          NA 2.061154e-09           NA
## (3,R2)           NA           NA 0.011439932 4.810889e-03 1.353580e-02
##              (3,R2)
## (1,R1)           NA
## (2,R1)           NA
## (3,R1) 0.030608107
## (1,R2) 0.004810889
## (2,R2) 0.013535796
## (3,R2)           NA
##
## Arrived at a reliable solution
```

Models with more states take longer to estimate because of the expansion of the state space and the number of transition rates connecting them. Hidden rate models further expand the state space. For example, adding a second rate category went from 4 transition rates to 14. In section 1.1 we left our parameters unconstrained. We estimated all transisions as independent and allowed for transitions from all states to any other state. However, we can constrain a model in corHMM in two different ways. The easiest way is to set the model to either "SYM" or "ER". This is what we've done for the HMM_3state model above. By setting model = "SYM", we have said that the transition rates between any two states are equal. If we set model = "ER", then we would have constrained all transition rates between states to be the same. Finally, if model = "ARD" (the default), then all transition rates are independently estimated. Although "ER" and "SYM are common restrictions, it is often more useful to manually restrict your model to match a biological hypothesis (which is described in the next section).

Interpreting the estimated rate matrix for this hidden Markov model is intimidating. But, the same principles of interpreting the transition rate matrices apply – that is, you still read rates from row to column. However, there is the added complexity of transitions among the different rate categories (as represented by R1 and R2). We have added a new plotting function called `plotMKmodel()` that will take the `corHMM` output and plot the underlying structure of model in discrete parts. In the following example, the first 2 panels show how observed states transition within each rate category, and the last panel shows transitions among the different rate classes:

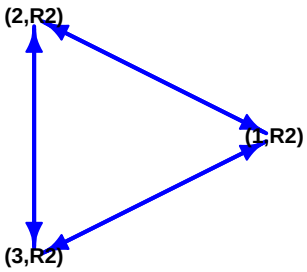
```
plotMKmodel(HMM_3state)
```

Rate Category 1 (R1)



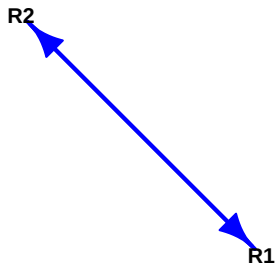
	(1,R1)	(2,R1)	(3,R1)
(1,R1)	--	100	0.01
(2,R1)	100	--	<0.01
(3,R1)	0.01	<0.01	--

Rate Category 2 (R2)



	(1,R2)	(2,R2)	(3,R2)
(1,R2)	--	<0.01	<0.01
(2,R2)	<0.01	--	0.01
(3,R2)	<0.01	0.01	--

Rate Category Transitions



	R1	R2
R1	--	0.03
R2	0.01	--

Section 2: How to make and interpret custom models

2.1: Creating and using custom rate matrices

2.1.1: One rate category

A custom rate matrix allows you to specify explicit hypotheses. For example, suppose you want to testwhether traits evolve in a certain order, testing for different rates of character evolution in different clades, or testing for the presence of hidden precursors before a state can evolve, then a custom model is the best way approach.

At its core, the purpose of a rate matrix (i.e., `rate.mat`) is to indicate to corHMM which parameters are being estimated. It specifies to `corHMM()` which rates in the matrix are being estimated and if any of them are expected to be identical. Let's start by using the `getStateMat4Dat()` function to get a generic `rate.mat` object:

```
LegendAndRateMat <- getStateMat4Dat(data)
RateMat <- LegendAndRateMat$rate.mat
RateMat
```

```
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   4
## (3)   0   3   0
```

Again, the numbers in this matrix are not rates, they are used to index the unique parameters to be estimated by `corHMM()`. Each distinct number is a parameter to be estimated independently from all others. Let's manually create the symmetric model we used in secion 1.2. In the symmetric model we want transitions *to* a state to be the same as *from* that state. This means that (1) → (2) & (2) → (1) are equal AND that (3) →(2) and (2) → (3) are equal. In other words, based on the `rate.mat` above, we want parameters 1 & 2 to be equal and we want parameters 3 & 4 to be equal:

```
pars2equal <- list(c(1,2), c(3,4))
StateMatA_constrained <- equateStateMatPars(RateMat, pars2equal)
StateMatA_constrained
```

```
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   2
## (3)   0   2   0
```

Here we used `equateStateMatPars()` for these purposes, where the first argument is *the rate matrix being modified* (i.e., `rate.mat` object) and second argument is *list of the parameters to be equated* to recreate the "SYM" model (i.e., `pars2equal` list). One thing to note is that you must have the appropriate number of rate categories. A user rate matrix will not be duplicated or changed by `corHMM()`. This custom model can only be used if we set rate.cat=1 since that is the appropriate number of rate categories. We can now provide this customized `rate.mat` to `corHMM()`:

```
MK_3state_customSYM <- corHMM(phy = phy, data = data, rate.cat = 1, rate.mat = StateMatA_constrained)
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
```

```
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
MK_3state_customSYM
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -44.36714 92.73429 92.94482        1   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)
## (1,R1)         NA 0.02569322         NA
## (2,R1) 0.02569322         NA 0.01969526
## (3,R1)         NA 0.01969526         NA
##
## Arrived at a reliable solution
```

2.1.2: Any number of rate categories

If you wanted to add hidden rate categories you need to know 2 things: (1) you need to know the dynamics *within* each rate category, and (2) transitions *among* the different rate classes. We begin by constructing two *within* rate category `rate.mat` objects. In the first category, R1, we assume all rates are equal. In the second rate categoy, R2, we suspect that for some species once they both estrus advertisement in a multimale mating system is an absorbing state – that is, the combination of estrus advertisement and multimale mating systems are never lost once they evolve:

```
RateCat1 <- getStateMat4Dat(data)$rate.mat # R1
RateCat1 <- equateStateMatPars(RateCat1, c(1:4))
RateCat1
```

```
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   1
## (3)   0   1   0
```

```
RateCat2 <- getStateMat4Dat(data)$rate.mat # R2
RateCat2 <- dropStateMatPars(RateCat2, 3)
RateCat2
```

```
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   3
## (3)   0   0   0
```

With respect to transitions *among* the different rate classes, we have implemented a separate matrix generator, `getRateCatMat()`. By default, this function will assume that all transitions among the specified number of rate classes occur independently. In our example, we will generate a matrix that specifies how transitions between R1 and R2 occur. Note that R1 and R2 could represent a biologically-relevant, but unmeasured factor, such as, say, temperate or tropical environments, island or mainland, presence or absence of a third trait. Basically, it is everything and anything that can influence the evolution of your observed characters.

For illustrative purposes, we will specify that the transition rate from R1 to R2 is the same as the rate from R2 to R1:

```
RateClassMat <- getRateCatMat(2) #
RateClassMat <- equateStateMatPars(RateClassMat, c(1,2))
RateClassMat
```

```
##    R1 R2
## R1  0  1
## R2  1  0
```

We now group all of our rate classes together in a list. The first element of the list corresponds to R1, the second to R2, etc.

```
StateMats <- list(RateCat1, RateCat2)
StateMats
```

```
## [[1]]
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   1
## (3)   0   1   0
##
## [[2]]
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   3
## (3)   0   0   0
```

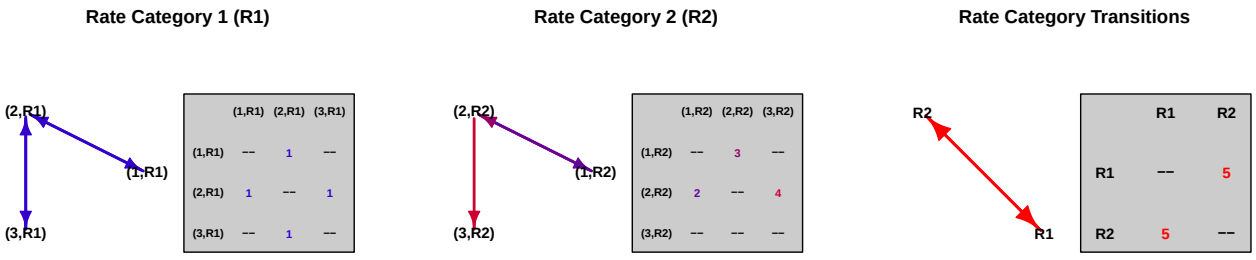
With that, we have all components necessary to create the full model using another function, `getFullMat()`, which requires that the first input be a list of the *within* rate class matrices and the second argument be the *among* rate class matrices.

```
FullMat <- getFullMat(StateMats, RateClassMat)
FullMat
```

```
##        (1,R1) (2,R1) (3,R1) (1,R2) (2,R2) (3,R2)
## (1,R1)      0      1      0      5      0      0
## (2,R1)      1      0      1      0      5      0
## (3,R1)      0      1      0      0      0      5
## (1,R2)      5      0      0      0      3      0
## (2,R2)      0      5      0      2      0      4
## (3,R2)      0      0      5      0      0      0
```

Even though we created this larger index matrix from individuals components we may not be sure it's exactly what we want. Like with visually the inferred model, we can use `plotMKmodel()` to take a look at the model setup *before* running the analysis:

```
plotMKmodel(pp = FullMat, rate.cat = 2, display = "row", text.scale = 0.7)
```



Since this is the model we intended on making, we can run `corHMM()` with our custom matrix,

```
HMM_3state_custom <- corHMM(phy = phy, data = data, rate.cat = 2, rate.mat = FullMat, node.states = "no
```

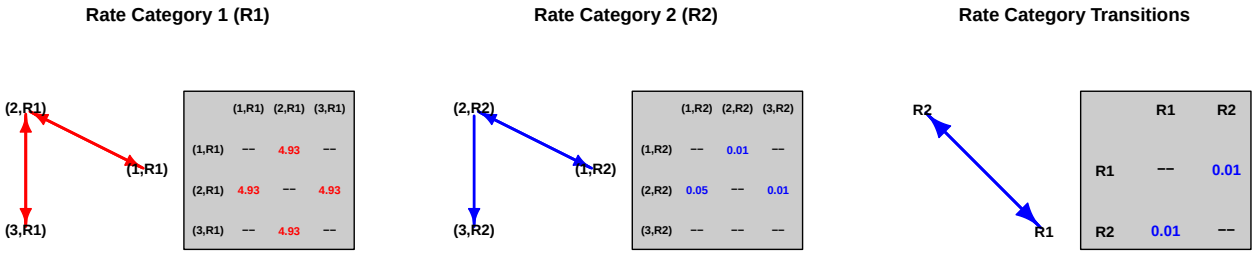
```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
```

```
HMM_3state_custom
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -42.23858 94.47717 95.58828        2   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)     (1,R2)      (2,R2)     (3,R2)
## (1,R1)         NA 4.93146841         NA 0.01011129          NA         NA
## (2,R1) 4.93146841         NA 4.93146841         NA 0.010111294         NA
## (3,R1)         NA 4.93146841         NA         NA          NA 0.01011129
## (1,R2) 0.01011129         NA         NA         NA 0.009100077         NA
## (2,R2)         NA 0.01011129         NA 0.04684400          NA 0.01248560
## (3,R2)         NA         NA 0.01011129         NA          NA         NA
##
## Arrived at a reliable solution
```

and then plot the resulting parameter estimates as before:

```
plotMKmodel(HMM_3state_custom, display = "row", text.scale = 0.7)
```



2.2: Biological examples

2.2.1: The precursor model

The precursor model of Marazzi et al. (2012) is a good example to start with, because it marks the beginning of HMMs being used in a comparative context. In that study, they were interested in locating putative evolutionary precursors of plant extrafloral nectaries (EFNs). Specifically, there were 2 states, presence (1) or absence (0) of EFNs, but that only species with an unobserved, hidden "precursor" trait could gain EFNs. Here we show how you could design the canonical precursor model in `corHMM` using custom rate matrices.

We will start by loading a simulated dataset of presence and absence of extrafloral nectaries a randomly generated birth-death tree:

```
phy <- read.tree("randomBD.tree")
load("simulatedData.Rsave")
head(Precur_Dat)
```

```
##      sp d
## s7   s7 0
## s14 s14 0
## s16 s16 0
## s17 s17 0
## s18 s18 1
## s21 s21 0
```

We will first generate a states only matrix using the input single binary trait data set:

```
Precur_LegendAndMat <- getStateMat4Dat(Precur_Dat)
Precur_LegendAndMat
```

```
## $legend
##   1   2
## "0" "1"
##
## $rate.mat
##     (1) (2)
## (1)   0   2
## (2)   1   0
```

Based on the legend, the absence of EFNs will be State 1 and the presence of EFNs will be State 2. The default states only matrix sets the rate of gain and rate of loss will differ. However, for a precursor model the transitions between the two observed states, 1 and 2, are modulated by a third, hidden trait, which we will call a precursor, but will be a state 1 in a different rate class. In other words, with the observation that a species lacking EFN's we do not know if they also have the precursor (i.e., 1, R2) or not (i.e., 1,R2). We do know, however, that under a precursor model that if we observe EFN, they must always also have the precursor trait, and so the presence of EFNs are always (2,R2). So, we will use rate class R2 as a means of transitioning between presence and absence of EFNs.

The first rate class, R1, will represent character changes in the presence of the precursor, which is not impossible without first gaining the "precursor". So, we will generate the default matrix, then drop all possible transitions from this matrix:

```
Precur_R1 <- Precur_LegendAndMat$rate.mat
Precur_R1 <- dropStateMatPars(Precur_R1, c(1,2))
Precur_R1
```

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

The second rate class, R2, will represent how our character changes in the presence of the precursor. In this rate class, we expect that species can either gain or lose EFNs at the same rate, but with State 1 representing no EFN but with precursor present:

```
Precur_R2 <- Precur_LegendAndMat$rate.mat
Precur_R2 <- equateStateMatPars(Precur_R2, c(1,2))
Precur_R2
```

```
##     (1) (2)
```

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

Finally, we set up a matrix for that governs the transitions *among* the rate classes:

```
RateClassMat <- getRateCatMat(2) #
RateClassMat <- equateStateMatPars(RateClassMat, c(1,2))
RateClassMat
```

```
##    R1 R2
## R1  0  1
## R2  1  0
```

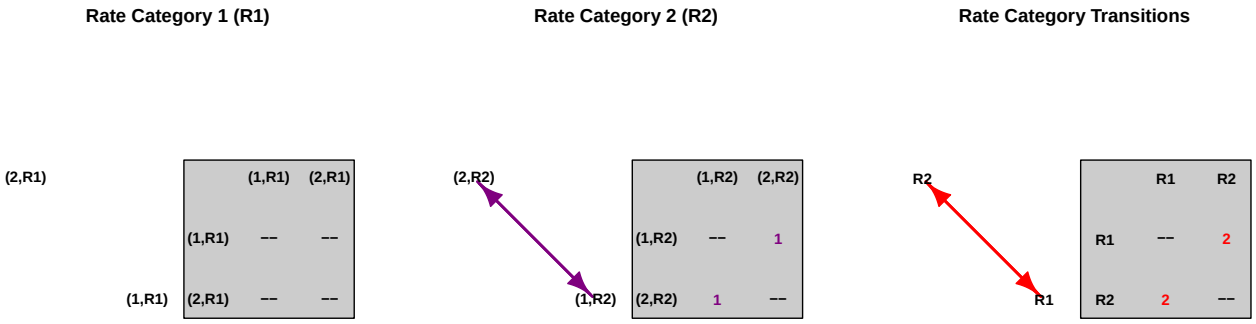
Putting them rate classes together we almost get the right model, but we need to remove one extra transition rate between that connects rate class R1 and R2 in the presence of EFNs, because, again, the precursor model assumes that EFNs can *only* be gained in rate class 1

```
Precur_FullMat <- getFullMat(list(Precur_R1, Precur_R2), RateClassMat)
Precur_FullMat[c(4,2), c(2,4)] <- 0
Precur_FullMat
```

```
##        (1,R1) (2,R1) (1,R2) (2,R2)
## (1,R1)      0      0      2      0
## (2,R1)      0      0      0      0
## (1,R2)      2      0      0      1
## (2,R2)      0      0      1      0
```

If you're unsure that the model is correct, you can plot the index matrix:

```
plotMKmodel(Precur_FullMat, 2, display = "row", text.scale = 0.7)
```



Since, it looks good we can now run `corHMM()` making sure to specify that we have 2 rate categories (or rate classes or hidden states - it's all the same).

```
Precur_res.corHMM <- corHMM(phy = phy, data = Precur_Dat, rate.cat = 2, rate.mat = Precur_FullMat, root
```

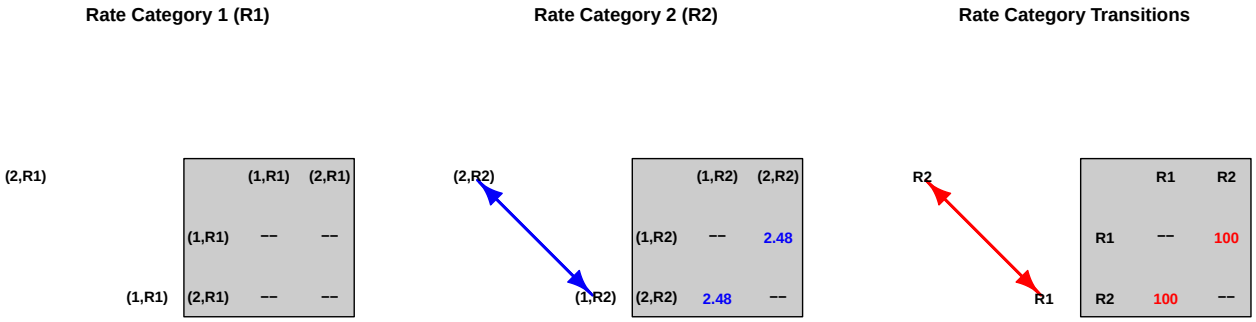
```
## State distribution in data:
## States:  1   2
## Counts:  57  43
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
Precur_res.corHMM
```

```
##
## Fit
##      -lnL      AIC     AICc Rate.cat ntax
##  -65.9401 135.8802 136.0039        2  100
##
```

```
## Rates
##          (1,R1) (2,R1)    (1,R2)   (2,R2)
## (1,R1)       NA     NA 99.997653       NA
## (2,R1)       NA     NA        NA       NA
## (1,R2) 99.99765     NA        NA 2.480293
## (2,R2)       NA     NA  2.480293       NA
##
## Arrived at a reliable solution
```

```
plotMKmodel(Precur_res.corHMM, display = "row", text.scale = 0.7)
```



2.2.2: Ordered habitat change

A lot of these new capabilities in `corHMM` were inspired by our current project examining the ancestral habitat during primary endosymbiosis. In our model, we have three possible habitats are marine, freshwater, and terrestrial. Our very large phylogeny of green plants contains many species with a diverse range of life histories. For example, cyanobacteria can move freely between all of these states, whereas some species may move between terrestrial and marine through freshwater. Some species may even move freely between aquatic states, but once they become terrestrial they are stuck there. In this section we will demonstrate how to create a custom hidden Markov model which satisfies all of these requirements.

To do this we will use a simulated a dataset that contains these 3 states:

```
load("simulatedData.Rsave")
head(MFT_dat)
```

```
##    sp           d
## 1  s7  Freshwater
## 2 s14      Marine
## 3 s16      Marine
## 4 s17 Terrestrial
## 5 s18 Terrestrial
## 6 s21      Marine
```

```
summary(as.factor(MFT_dat[,2])) # how many of each state do we have?
```

```
##  Freshwater      Marine Terrestrial
##           7          14          79
```

As before, start off by getting a legend and rate matrix from this dataset:

```
MFT_LegendAndRate <- getStateMat4Dat(MFT_dat)
MFT_LegendAndRate
```

```
## $legend
##             1             2             3
```

```
##  "Freshwater"      "Marine" "Terrestrial"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   3   5
## (2)   1   0   6
## (3)   2   4   0
```

Here, freshwater habitat will be State 1, marine habitat will be State 2,and terrestrial habitat will be State 3. Now, we need to create 3 different rate classes that are consistent with our hypotheses of how habitat changes occurs. We'll say that rate class R1 is one in which lineages cannot leave a terrestrial habitat, rate class R2 will allow linneages to transition between marine and terrestrial *only* through freshwater, and rate class R3 will be unrestricted movement between the habitats.

For R1 we need terrestrial to be an absorbing state, meaning once terrestriality evolves it is not lost. Since 1 = Freshwater, 2 = Marine, and 3 = Terrestrial, that means removing from (3) to (1) and from (3) to (2).

```
MFT_R1 <- dropStateMatPars(MFT_LegendAndRate$rate.mat, c(2,4))
MFT_R1
```

```
##     (1) (2) (3)
## (1)   0   2   3
## (2)   1   0   4
## (3)   0   0   0
```

For R2, we need to disallow transitions between terrestrial and marine. We disallow the positions (1,3) and (3,1) in the rate matrix. In this case any lineage can move into freshwater and move out of freshwater, but they are not allowed to transition directly between terrestrial and marine habitats:

```
MFT_R2 <- dropStateMatPars(MFT_LegendAndRate$rate.mat, c(4,6))
MFT_R2
```

```
##     (1) (2) (3)
## (1)   0   3   4
## (2)   1   0   0
## (3)   2   0   0
```

For R3, this is our free-moving matrix, which is simply the default matrix provided by getStateMat4Dat:

```
MFT_R3 <- MFT_LegendAndRate$rate.mat
MFT_R3
```

```
##     (1) (2) (3)
## (1)   0   3   5
## (2)   1   0   6
## (3)   2   4   0
```

Let's put all these matrices in a list,

```
MFT_ObsStateClasses <- list(MFT_R1, MFT_R2, MFT_R3)
```

and since we only have 100 species let's constrain our parameters a bit further and say transitions between rate classes occur at the same rate:

```
MFT_RateClassMat <- getRateCatMat(3) # we have 3 rate classes
MFT_RateClassMat <- equateStateMatPars(MFT_RateClassMat, 1:6)
```

And we put it all together into a corHMM compatible rate.mat.

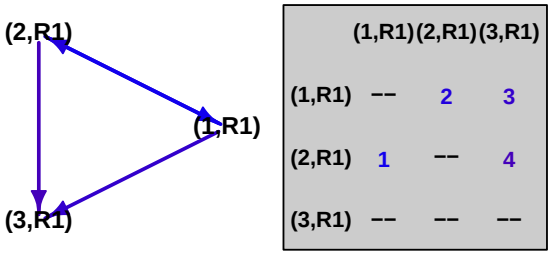
```
MFT_FullMat <- getFullMat(MFT_ObsStateClasses, MFT_RateClassMat)
MFT_FullMat
```

```
##         (1,R1) (2,R1) (3,R1) (1,R2) (2,R2) (3,R2) (1,R3) (2,R3) (3,R3)
## (1,R1)      0      2      3     15      0      0     15      0      0
## (2,R1)      1      0      4      0     15      0      0     15      0
## (3,R1)      0      0      0      0      0     15      0      0     15
## (1,R2)     15      0      0      0      7      8     15      0      0
## (2,R2)      0     15      0      5      0      0      0     15      0
## (3,R2)      0      0     15      6      0      0      0      0     15
## (1,R3)     15      0      0     15      0      0      0     11     13
## (2,R3)      0     15      0      0     15      0      9      0     14
## (3,R3)      0      0     15      0      0     15     10     12      0
```

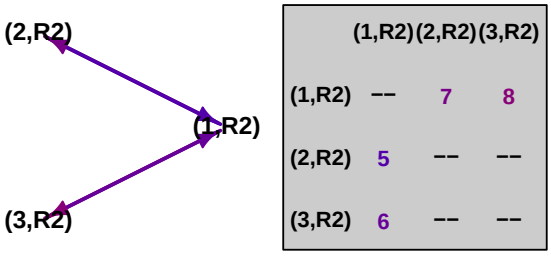
That's kind of difficult to interpret, so let's plot it out and see if it's what we wanted:

```
plotMKmodel(pp = MFT_FullMat, rate.cat = 3, display = "square", text.scale = 0.9)
```

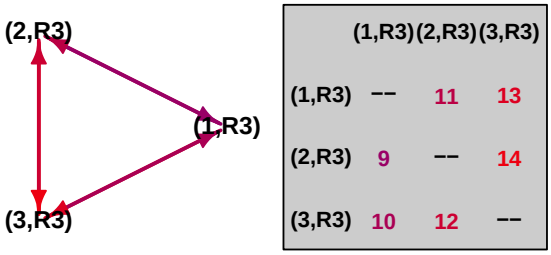
Rate Category 1 (R1)



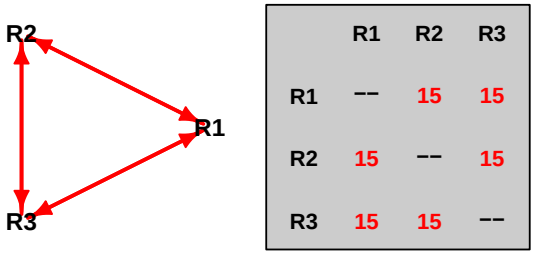
Rate Category 2 (R2)



Rate Category 3 (R3)



Rate Category Transitions



To run this model, we would only need to specify the data, the phylogeny, this matrix, and that this matrix has 3 rate categories.

```
MFT_res.corHMM <- corHMM(phy = phy, data = MFT_dat, rate.cat = 3, rate.mat = MFT_FullMat, node.states =
```

```
## State distribution in data:
## States:  1   2   3
## Counts:  7   14  79
## Beginning thorough optimization search -- performing 0 random restarts
```

```
MFT_res.corHMM
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -56.62103 143.2421 148.9563        3  100
##
## Rates
##              (1,R1)    (2,R1)       (3,R1)    (1,R2)   (2,R2)       (3,R2)
## (1,R1)           NA 87.828656 2.061154e-09 4.5326985       NA           NA
## (2,R1) 2.062959e-09        NA 2.061154e-09        NA 4.532698           NA
## (3,R1)           NA        NA           NA        NA       NA 4.532698e+00
## (1,R2) 4.532698e+00        NA           NA        NA 1.799977 2.061154e-09
## (2,R2)           NA  4.532698           NA 0.3457312       NA           NA
## (3,R2)           NA        NA 4.532698e+00 1.4308419       NA           NA
## (1,R3) 4.532698e+00        NA           NA 4.5326985       NA           NA
## (2,R3)           NA  4.532698           NA        NA 4.532698           NA
## (3,R3)           NA        NA 4.532698e+00        NA       NA 4.532698e+00
##              (1,R3)       (2,R3)       (3,R3)
## (1,R1) 4.532698e+00           NA           NA
## (2,R1)           NA 4.532698e+00           NA
## (3,R1)           NA           NA 4.532698e+00
## (1,R2) 4.532698e+00           NA           NA
## (2,R2)           NA 4.532698e+00           NA
## (3,R2)           NA           NA 4.532698e+00
## (1,R3)           NA 2.061154e-09 2.061154e-09
## (2,R3) 2.061154e-09           NA 1.523958e+01
## (3,R3) 2.061154e-09 2.061154e-09           NA
##
## Arrived at a reliable solution
```

2.2.3: Ontological relationship of multiple characters

Lets say we had a dataset with multiple characters: presence or absence of limbs, presence or absence of fingers, corporeal or incorporeal form. It could look something like this:

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
Limbs <- c("Limbs", "noLimbs")[data[,2]+1]
Fings <- vector("numeric", length(phy$tip.label))
Fings[which(Limbs == "Limbs")] <- round(runif(length(which(Limbs == "Limbs")), 0, 1))
Corpo <- rep("corporeal", length(phy$tip.label))
Ont_Dat <- data.frame(sp = phy$tip.label, limbs = Limbs, fings = Fings, corp = Corpo)
head(Ont_Dat)
```

```
##                     sp   limbs fings      corp
## 1          Homo_sapiens noLimbs     0 corporeal
## 2          Pan_paniscus   Limbs     1 corporeal
```

```
## 3       Pan_troglodytes   Limbs     1 corporeal
## 4       Gorilla_gorilla   Limbs     0 corporeal
## 5        Pongo_pygmaeus   Limbs     0 corporeal
## 6 Pongo_pygmaeus_abelii   Limbs     0 corporeal
```

Previously, the user would have had to convert this dataset into something `rayDISC()` could use, such as taking all possible unique combinations and creating a multistate character. Again, `corHMM()` will internally do this for you:

```
Ont_LegendAndMat <- getStateMat4Dat(Ont_Dat)
Ont_LegendAndMat
```

```
## $legend
##                          1                          2
##   "Limbs & 0 & corporeal"   "Limbs & 1 & corporeal"
##                          3                         NA
## "noLimbs & 0 & corporeal" "noLimbs & 1 & corporeal"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   3   4
## (2)   1   0   0
## (3)   2   0   0
```

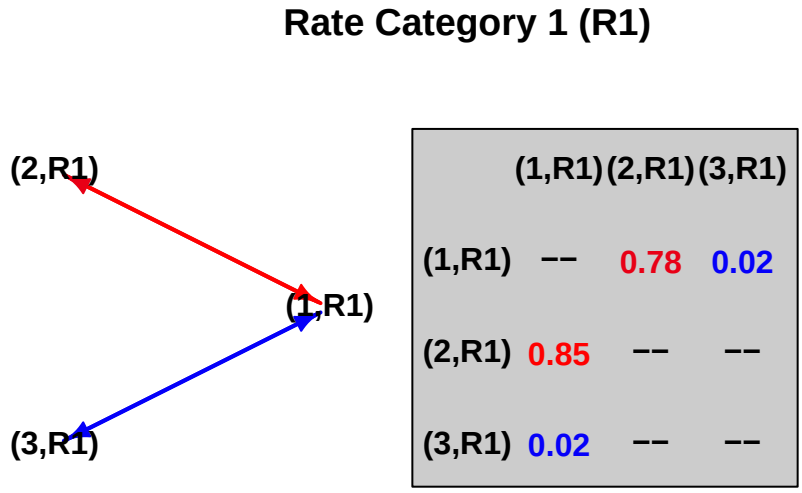
Even though there were 3 binary characters (meaning 8 possible states), only 3 combinations were actually observed. This is because all of the species were corporeal and thus that state didn't factor into the matrix structure. The next thing to notice is that one of the potential states (No Limbs, Yes Fingers) is not present in the dataset and thus not included in the model. Finally, dual transitions have been removed. The transition from 3 (No Limbs, No Fingers) to 2 (Yes Limbs, Yes Fingers) is not allowed.

```
Ont_res.corHMM <- corHMM(phy = phy, data = Ont_Dat, rate.cat = 1, rate.mat = Ont_LegendAndMat$rate.mat,
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##                          1                          2
##   "Limbs & 0 & corporeal"   "Limbs & 1 & corporeal"
##                          3                         NA
## "noLimbs & 0 & corporeal" "noLimbs & 1 & corporeal"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  20  19  21
## Beginning thorough optimization search -- performing 0 random restarts
```

Plotting the results:

```
plotMKmodel(Ont_res.corHMM)
```



Note that hidden states can be added to this model rather easily by following the examples above.

2.3: Fixing states at nodes

We added the ability to fix any or all nodes in the input phylogeny while estimating a model. This new feature was inspired by a request from Scott Edwards, who was interested in the range of rates of flight gain and loss will result in the highest probability of a volant ancestor to flightless lineages. He ran a series of ancestral state reconstructions under a range of rates of gain and loss of flight, focusing on a single ancestor at a time (since each ancestor will have a slightly different set of parameters), and recording the probability of a volant ancestor. These analyses are included in the supplemental of Sackton et al. (2019).

In this updated version of `corHMM` a user can fix anywhere from a single node in a tree to an entire reconstruction from, say parsimony, and estimate the transitions rate. One can even obtain the likelihood of a reconstruction based on a fixed set of rates. To demonstrate, let's start by running a simple analysis of a binary character, but fixing the state of a single node in the the primate tree. Specifically, we are going to fix the most recent common ancestor of *Gorilla gorilla* and *Homo sapiens* as exhibiting estrus advertisement (i.e., state 1). The first step is to determine the the indices for each state:

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
getStateMat4Dat(data[,c(1,2)])
```

```
## $legend
##   1   2
## "0" "1"
##
## $rate.mat
##     (1) (2)
## (1)   0   2
## (2)   1   0
```

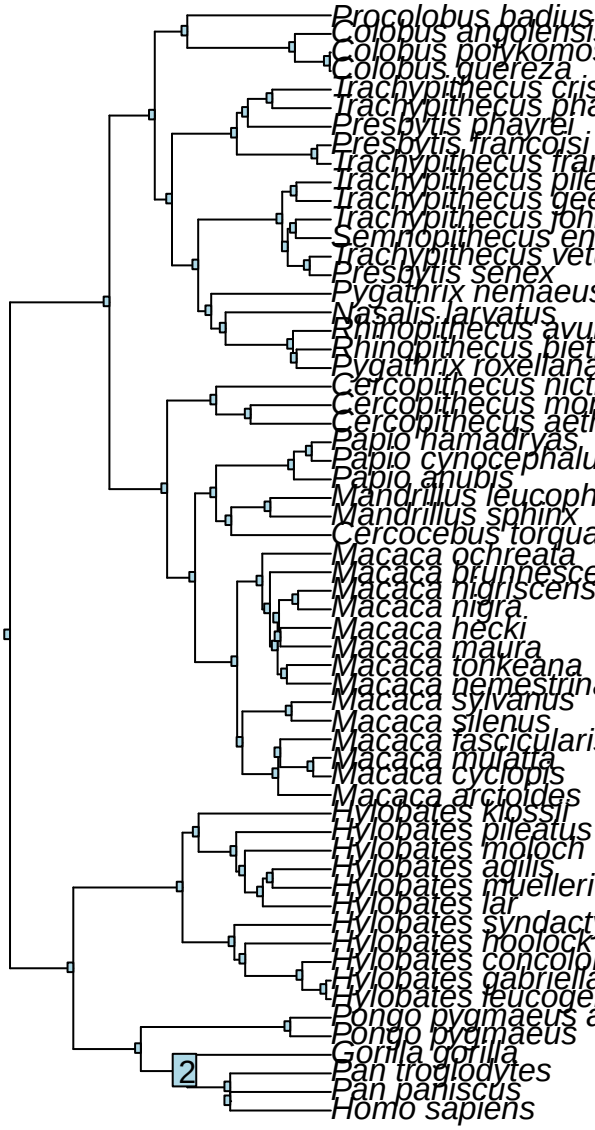
So, here the index 2 represents the presence of estrus advertisement in the model. The next step is create a vector of node states. We start by generating a string of `NA` of length equal to the number of nodes plus the number of tips in the tree. The `NA` simply tells `corHMM()` to ignore as these are nodes that are not fixed. We then have to determine which node is the MRCA of *Gorilla gorilla* and *Homo sapiens*:

```
label.vector <- rep(NA, Ntip(phy) + Nnode(phy))
homo_gorilla <- getMRCA(phy,tip=c("Homo_sapiens", "Gorilla_gorilla"))
homo_gorilla
```

```
## [1] 64
```

The node number should be 64. To set the state of the node, simply replace the `NA` with the state of interest in the `label.vector`, in this case with a 2, in the 64th element. To set the node labels, we will clip off the first 60 elements, as these represent the states of the tips, which we set differently:

```
label.vector[homo_gorilla] <- 2
phy$node.label <- label.vector[-c(1:Ntip(phy))]
```



Plotting the tree can help to double-check whether the right node was fixed:

From here simply input the `tree` object in `corHMM()` as normal, but the option fix.nodes needs to be set to `TRUE`:

```
## State distribution in data:
## States:  1   2
## Counts:  39  21
```

```
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

We can then compare the fit of this model, with one where the same node is fixed to lacking estrus advertisement:

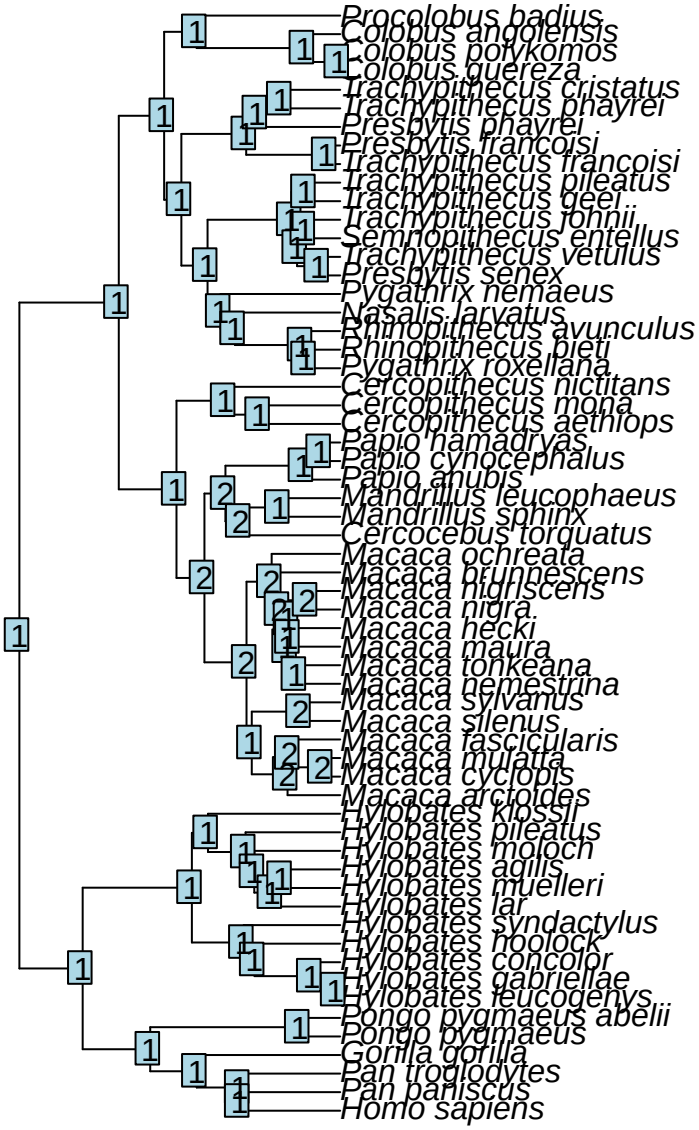
```
label.vector[homo_gorilla] <- 1
phy$node.label <- label.vector[-c(1:Ntip(phy))]
fix.node64.noestrus <- corHMM(phy, data[,c(1,2)], model="ER", rate.cat=1, fixed.nodes=TRUE)
```

```
## State distribution in data:
## States:  1   2
## Counts:  39  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

A model where the MRCA of *Gorilla gorilla* and *Homo sapiens* is assumed to have exhibited estrus advertisement requires higher rates, and produces a substantially worse likelihood, than if we assume the MRCA lacked estrus advertisement.

It is also possible to estimate transition rates where all nodes are fixed in the tree. For example, what if one wanted to examine the fit of a model where all nodes are fixed to according to a maximum parsimony reconstruction. Here we will use `phangorn` for these purposes. However, with `phangorn` there is the burden of dealing with their unique `phyDat` format. To deal with this, we implemented a function, `ConvertPhangornReconstructions()` that will convert the `phyDat` formatted output into something we can modify and input into `corHMM()`. We can take the `mpr.recon` object from `phangorn` and convert the output as a vector and add them as node states in the phylogeny:

```
library(phangorn)
data.sort <- data.frame(data[,2], row.names=data[,1])
data.sort <- data.sort[phy$tip.label,]
dat<-as.matrix(data.sort)
rownames(dat) <- phy$tip.label
dat<-phyDat(dat,type="USER", levels=c("0","1"))
mpr.recon <- ancestral.pars(phy, dat, type = c("MPR"))
mpr.recon.converted <- ConvertPhangornReconstructions(mpr.recon)
phy$node.label <- mpr.recon.converted[(Ntip(phy)+1):length(mpr.recon.converted)]
```



Plotting the tree shows the parsimony reconstruction:

Next input the tree into `corHMM()` and obtain a rate estimate for this reconstruction:

```
fixed.parsimony.recon <- corHMM(phy, data[,c(1,2)], model="ER", rate.cat=1, fixed.nodes=TRUE)
```

```
## State distribution in data:
## States:  1   2
## Counts:  39  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

Interestingly, the parsimony reconstruction suggests a lot more change than if we estimated the states from the model itself.

Finally, if the model contains hidden states the user simply needs to fix state of the node based on the observed state *only*. Remember, we cannot actually observe hidden states, so we must treat the state of the node as ambiguous across all possible rate classes like we would a tip. Let's run a quick example where we fix the MRCA of *Gorilla gorilla* and *Homo sapiens* as lacking estrus advertisement:

```
label.vector <- rep(NA, Ntip(phy) + Nnode(phy))
homo_gorilla <- getMRCA(phy,tip=c("Homo_sapiens", "Gorilla_gorilla"))
```

```
label.vector[homo_gorilla] <- 1
phy$node.label <- label.vector[-c(1:Ntip(phy))]
fix.node64.noestrus <- corHMM(phy, data[,c(1,2)], model="ARD", rate.cat=2, fixed.nodes=TRUE)
```

```
## State distribution in data:
## States:  1   2
## Counts:  39  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

Now, if we print out the line corresponding to our fixed node,

```
fix.node64.noestrus$states[homo_gorilla-Ntip(phy),]
```

```
##    (1,R1)    (2,R1)    (1,R2)    (2,R2)
## 0.8750823 0.0000000 0.1249177 0.0000000
```

there should be some uncertainty as to whether the absence of estrus advertisement is in R1 or R2. The total probability, however, of the node being in observed state 1 should sum to 1:

```
sum(fix.node64.noestrus$states[homo_gorilla-Ntip(phy),])
```

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

References

Beaulieu J.M., B.C. O'Meara, and M.J. Donoghue. 2013. Identifying hidden rate changes in the evolution of a binary morphologicalcharacter: the evolution of plant habit in campanulid angiosperms. Systematic Biology 62:725-737.

Pagel, M. 1994. Detecting correlated evolution on phylogenies: a general method for the comparative analysis of discrete characters. Proceedings of the Royal Society, B. 255:37-45.

Pupko, T., I. Pe'er, R. Shamir, and D. Graur. 2000. A fast algorithm for joint reconstruction of ancestral amino-acid sequences. Molecular Biology and Evolution 17:890-896.

Sackton, T.B., P. Grayson, A. Cloutier, Z. Hu, J.S. Liu, N.E. Wheeler, P.P. Gardner, J.A. Clarke, A.J. Baker, M. Clamp, and S.V. Edwards. 2019. Convergent regulatory evolution and loss of flight in paleognathous birds. Science 364:74-78.