Estimating phylogenetic trees with phangorn (Version 0.99-2)

Klaus P. Schliep[*]

July 20, 2009

1 Introduction

These notes should enable the user to estimate phylogenetic trees from alignment data with with different methods using the *phangorn* package. For more background on all the methods see e.g. [2, 7]. This document illustrates some of the *phangorn* features to estimate phylogenetic trees using different reconstruction methods. Small adaptations to the scripts in 6 should enable the user to perform phylogenetic analysis.

2 Getting started

The first thing we have to do is to read in an alignment. Unfortunately there exists many different file formats alignments can be stored in. The function `read.phyDat` is used to read in an alignment. There are several functions to read in alignments depending on the format of the dataset (nexus, phylip, fasta) and the kind of data (amino acid or nucleotides) in the *ape* package [3] and *phangorn*. The function `read.phyDat` calls these other functions. For the specific parameter settings available look in the help files of the function `read.dna` (for phylip, fasta, clustal format), `read.nexus.data` for nexus files. For amino acid data additional `read.aa` is called. We start our analysis the *phangorn* package and then load in an alignment.

```
> library(phangorn)
> primates = read.phyDat("primates.dna", format = "phylip",
+     type = "DNA")
```

[*] mailto:k.p.schliep@massey.ac.nz

3 Distance based methods

After reading in the alignment we can build a first tree with distance based methods. The function dist.dna from the ape package computes distances for many DNA substitution models. To use the function dist.dna we have to transform the data to class DNAbin. For amino acids the function dist.ml offers common substitution models (”WAG”, ”Dayhoff”, ”JTT” and ”LG”). After constructing a distance matrix we reconstruct a rooted tree with UPGMA and alternatively an unrooted tree using Neighbor Joining [5, 6].

```
> dm = dist.dna(as.DNAbin(primates))
> treeUPGMA = upgma(dm)
> treeNJ = NJ(dm)
```

We can use the plot the trees treeUPGMA and treeNJ (figure 1) with the commands:

```
> par(mfrow = c(1, 2), mar = c(1, 1, 4, 1))
> plot(treeUPGMA, main = "UPGMA")
> plot(treeNJ, "unrooted", main = "NJ")
```

Distance based methods are very fast and we will use the UPGMA and NJ tree as starting trees for the maximum parsimony and maximum likelihood analysis.

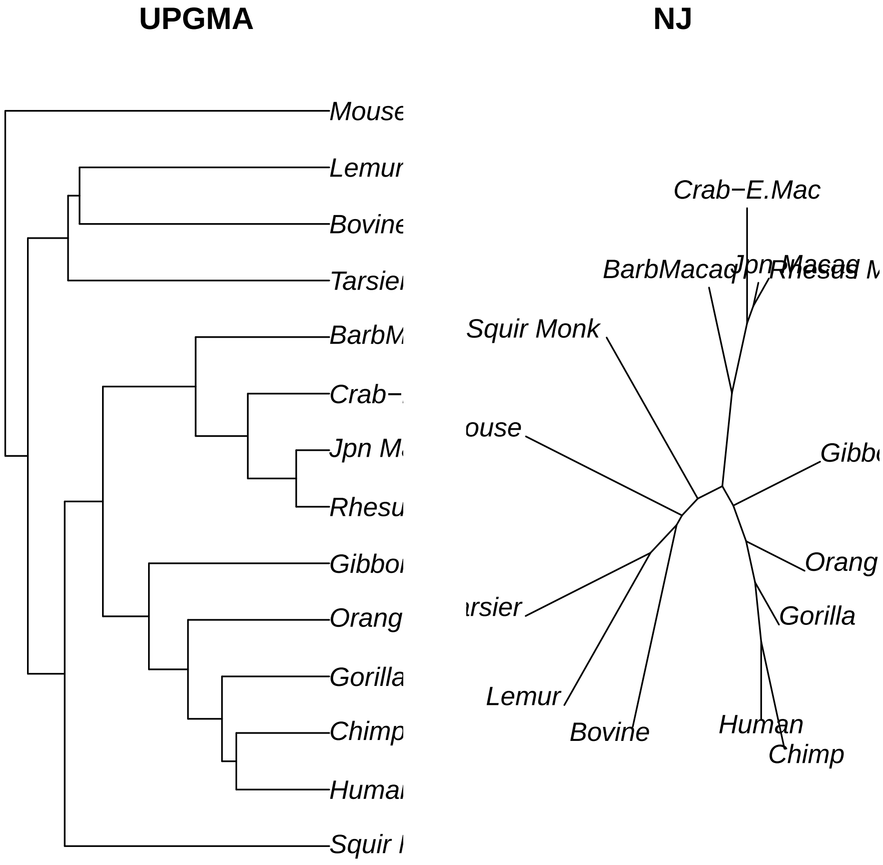
4 Parsimony

The function parsimony returns the parsimony score, that is the number of changes which are at least necessary to describe the data for a given tree. We can compare the parsimony score or the two trees we computed so far:

```
> parsimony(treeUPGMA, primates)
[1] 751
> parsimony(treeNJ, primates)
[1] 746
```

We can use the function optim.parsimony performs tree rearrangements to find trees with a lower parsimony score. So far the only tree rearrangement implemented is nearest-neighbor interchanges (NNI).

```
> treePars = optim.parsimony(treeUPGMA, primates)
optimize topology:  751 --> 746
optimize topology:  746 --> 746
Final p-score 746 after  1 nni operations
> parsimony(treePars, primates)
[1] 746
```


UPGMA

NJ

Figure 1: Rooted UPGMA tree and unrooted NJ tree

5 Maximum likelihood

The last method we will describe in this vignette is Maximum Likelihood (ML) as introduced by Felsenstein [1]. We can easily compute the likelihood for a tree given the data

```
> fit = pml(treeNJ, data = primates)
> fit
 loglikelihood: -3077.846

unconstrained loglikelihood: -1230.335

Rate matrix:
  a c g t
a 0 1 1 1
c 1 0 1 1
g 1 1 0 1
t 1 1 1 0

Base frequencies:
0.25 0.25 0.25 0.25
```

The function pml returns an object of class pml. This object contains the data, the tree and many different parameters of the model like the likelihood etc. There are many generic functions for the class pml available.

The object fit just estimated the likelihood for the tree it got supplied, but the branch length are not optimized for for the Jukes-Cantor model yet, which can be done with the function optim.pml.

```
> fitJC = optim.pml(fit, TRUE)
> logLik(fitJC)
```

With the default values `pml` will estimate a Jukes-Cantor model. The function `update.pml` allows to change parameters. We will change the model to the GTR + $\Gamma(4)$ + I model and then optimize all the parameters.

```
> fitGTR = update(fit, k = 4, inv = 0.2)
> fitGTR = optim.pml(fitGTR, TRUE, TRUE, TRUE, TRUE, TRUE)


> fitGTR
 loglikelihood: -2609.589

unconstrained loglikelihood: -1230.335
```

```
Proportion of invariant sites: 0.006045091
Discrete gamma model
Number of rate categories: 4
Shape parameter: 3.175621

Rate matrix:
           a          c        g          t
a  0.0000000  0.6234389 32.36013  0.3867576
c  0.6234389  0.0000000  0.00000 13.8337603
g 32.3601271  0.0000000  0.00000  1.0000000
t  0.3867576 13.8337603  1.00000  0.0000000

Base frequencies:
0.3918068 0.3795443 0.04024686 0.1884020
```

We can compare the trees for the JC and GTR + $\Gamma(4)$ + I model with the AIC

```
> AIC(fitGTR)
[1] 5293.178
> AIC(fitJC)
[1] 6186.59
```

or the Shimodaira-Hasegawa test.

```
> SH.test(fitGTR, fitJC)
     Trees      ln L Diff ln L p-value
[1,]     1 -2609.589    0.0000   0.498
[2,]     2 -3068.295  458.7061   0.000
```

6 Appendix: Standard scripts for nucleotide or amino acid analysis

Here we provide two standard scripts which can be adapted for the most common tasks. Most likely the arguments for read.phyDat have to be adapted to accommodate your file format. The bootstrap analysis can be computational demanding.

```
> file = "myfile"
> dat = read.phyDat(file)
> dm = dist.ml(dat)
> tree = NJ(dm)
> fitNJ = pml(tree, dat, k = 4, inv = 0.2)
> fit = optim.pml(fitNJ, TRUE, TRUE, TRUE, TRUE, TRUE)
> fit
```

You can specify different several models build in which you can specify ”WAG”, ”JTT”, ”Dayhoff”, ”LG”. Optimising the rate matrix for amino acids is possible, but would take a long, a very long time. So make sure to set optBf=FALSE and optQ=FALSE in the function `optim.pml`, which is also the default.

```
> file = "myfile"
> dat = read.phyDat(file, type = "AA")
> dm = dist.ml(dat, model = "JTT")
> tree = NJ(dm)
> fitNJ = pml(tree, dat, model = "JTT", k = 4, inv = 0.2)
> fit = optim.pml(fitNJ, optNni = TRUE, optInv = TRUE,
+     optGamma = TRUE)
> fit
```

References

[1] Joseph Felsenstein. Evolutionary trees from dna sequences: a maxumum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[2] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, 2004.

[3] E. Paradis, J. Claude, and K. Strimmer. Ape: Analyses of phylogenetics and evolution in r language. *Bioinformatics*, 20(2):289–290, 2004.

[4] Emmanuel Paradis. *Analysis of Phylogenetics and Evolution with R*. Springer, New York, 2006.

[5] N. Saitou and M. Nei. The neighbor-joining method - a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[6] J. A. Studier and K. J. Keppler. A note on the neighbor-joining algorithm of saitou and nei. *Molecular Biology and Evolution*, 5(6):729–731, 1988.

[7] Ziheng Yang. *Computational Molecular evolution*. Oxford University Press, Oxford, 2006.

7 Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 2.9.1 (2009-06-26), `i386-pc-mingw32`
- Locale: `LC_COLLATE=English_New Zealand.1252;LC_CTYPE=English_New Zealand.1252;LC_MON`

- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: ape 2.3-1, phangorn 0.99-2, quadprog 1.4-11
- Loaded via a namespace (and not attached): gee 4.13-13, grid 2.9.1, lattice 0.17-25, nlme 3.1-92