Phylogenetic Inference using RevBayes

Gene tree-Species tree reconstruction

1 Overview: Gene tree-species tree models

Ever since Zuckerkandl and Pauling (Zuckerkandl and Pauling, 1965), people have recognised that phylogenies reconstructed from homologous gene sequences could differ from species phylogenies. As molecular sequences accumulated, the link between gene trees and species trees started to be modelled. The first models were based on parsimony, and aimed for instance at reconciling a gene tree with a species tree by minimizing the number of events of gene duplication and gene loss. In the past dozen years, probabilistic models have been proposed to reconstruct gene trees and species trees in a rigorous statistical framework. Models and algorithms have quickly grown in complexity, to model biological processes with increasing realism, to accommodate several processes at the same time, or to handle genome-scale data sets. In this overview we will not detail these models, and we invite the interested reader to take a look at recent reviews (e.g. (Szöllősi et al., 2014)).

1.1 Processes of discord

There are several reasons why a gene tree may differ from a species tree. Of course, a gene tree may differ from the species tree just because a mistake was made during the analysis of the gene sequences, at any point in a pipeline going from the sequencing itself to the tree reconstruction. Such a mistake would produce an incorrect gene tree. Here we do not mean this kind of discord, but rather discord that has comes from a real biological process that builds true gene histories that differ from true species histories. These processes include gene duplication, gene loss, gene transfer (used loosely here to also include reticulation, hybridization between species), and incomplete lineage sorting (Fig. 1). Incomplete lineage sorting will be discussed in more details in the following subsection.

Fig. 1 suggests that for all processes the gene tree can be seen as the product of a branching process operating inside the species tree. As a consequence, all processes are modelled as some type of birth-death process. For duplication/loss models, birth correspond to gene duplication events, and death to gene loss events. Transfers can be added to the model by introducing another type of birth, with a child lineage appearing in another branch of the species tree. Incomplete lineage sorting is also modelled with a birth-death type of model, the coalescent. All these models can be made heterogeneous, by allowing different sets of parameters for different branches of the species tree. This is useful to model differences in rates of duplication, loss or transfer among species, or to model different effective population sizes in a species tree. In RevBayes so far only models of incomplete lineage sorting have been implemented (models of duplication and loss and transfer will soon be added). Thanks to RevBayes modular design, there is quite a lot of flexibility in specifying the model, for instance by allowing different parameters to different branches of the species tree, and the gene tree-species tree model could be combined to other types of models, for instance models of trait evolution.

1.2 Gene tree discordance is a problem for species tree reconstruction

There are several approaches to species tree reconstruction: concatenation and supertree approaches, which have been used for quite some time now, and more recently methods that rely on gene tree-species tree models.

1. Concatenation simply consists in taking all gene alignments, concatenating them into one super

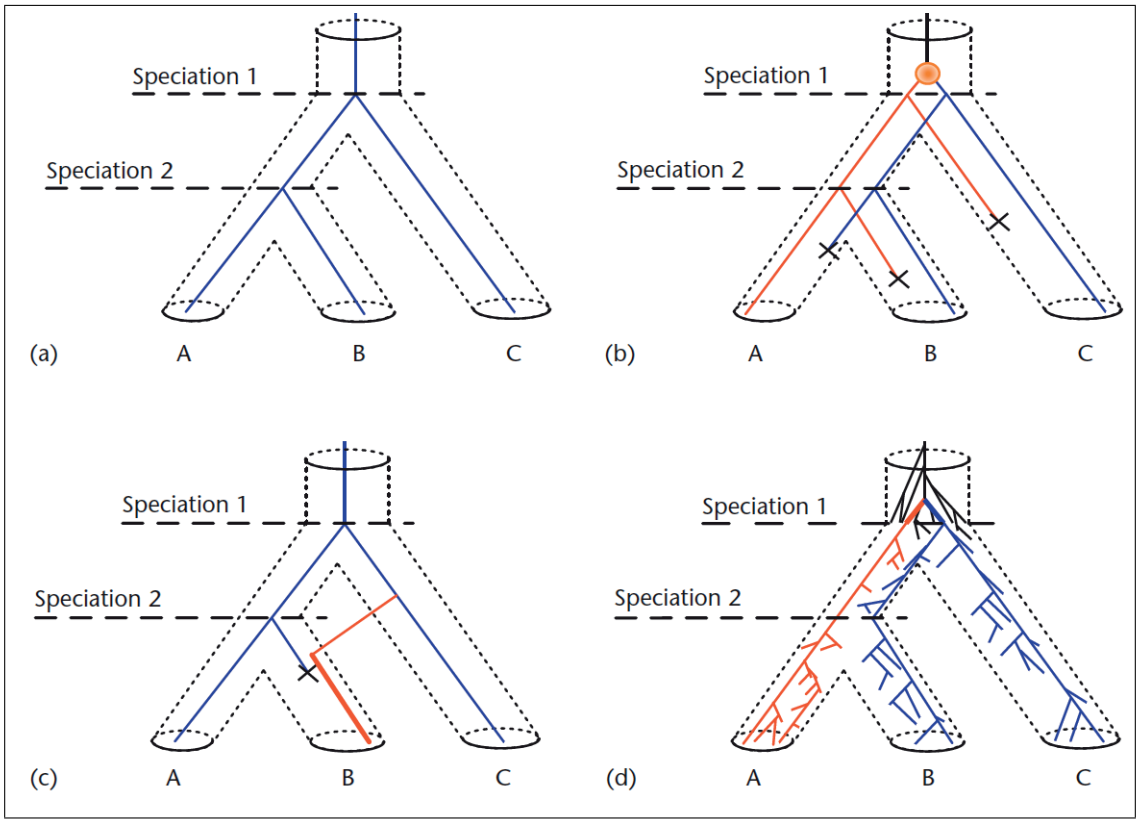

Figure 1: The processes of discord. The species tree is represented as a tubular structure. Gene trees are blue and red lines running along the species trees. a) A gene tree that perfectly matches the species tree. b) The gene tree and the species tree differ because of gene duplications and losses. c) The gene tree and the species tree differ because of gene transfer and gene loss. d) The gene tree and the species tree differ because of incomplete lineage sorting. [Replicated from Fig. 2 in Boussau (2009).]

alignment, and then analyzing it as if it were a single gene sequence. Its main assumption is therefore that all sites of all genes have evolved according to the same species tree. This assumption is not correct because all the processes of discord presented above conspire to make gene trees different from the species tree. In practice, this matters: for instance, one can prove that in the presence of incomplete lineage sorting, in some particular area of the parameter space, concatenation will return an incorrect species tree. Another example might be found in prokaryotic phylogenetics, where the quest for a tree of life has been very frustrating, to the point that many doubt that one could find a meaningful species tree representing vertical descent. Recent models incorporating lateral gene transfer allow tackling this question in a more principled way.

2. Supertree approaches differ from concatenation notably by discarding sequence information once individual gene trees have been built. They combine individual gene trees to obtain a species tree. Most supertree methods are not based on an explicit model of the processes causing discordance between gene trees and species tree (although there are exceptions, notably modelling incomplete lineage sorting, see below). Instead, they aim at finding a tree that would best describe the distribution of gene trees, according to some fairly arbitrary criterion. In practice, these methods have been found to provide reasonable results in many cases, but in simulations are less accurate than concatenation.

3. Methods that rely on gene tree-species tree models appear very promising as they explicitly model

the processes of discord, and can be combined with a model of sequence evolution, models of the co-evolution between gene trees, models of trait evolution...

1.3 Modelling incomplete lineage sorting: the multispecies coalescent

Incomplete lineage sorting is a population-level process. In a species, at a given time, there are several alleles for a given locus in the genome. These alleles have their own history, they diverged from each other at various times in the past. This history can differ from the species history, because several alleles can persist through a speciation event, and because, short of selective effects, the sorting of alleles during a speciation event is random and can result in a tree that differs from the species tree (Fig. 1d). In all cases, incongruence between the gene tree and the species tree occurs when alleles persist over the course of several speciation events. When reconstructing a gene tree, one therefore gets the history of the alleles that have been sampled (at best), not necessarily the history of the species.

In 2003, Rannala and Yang proposed a powerful way to model the sorting of alleles along a phylogeny of several species (Rannala and Yang, 2003), the multispecies coalescent (Fig. 2). This model is at the origin of most model-based approaches to reconstruct gene and species trees (Edwards, Liu and Pearl, 2007; Heled and Drummond, 2010). The multispecies coalescent appropriately models the evolution of a population of alleles along a species tree. Along the species tree, it allows different branch lengths, in units of time, and also allows different effective population sizes. Computing the probability of a gene tree given a species tree and other parameters is quite easy. Bascially it works by cutting the gene tree into independent species-specific subtrees, computing probabilities for each of those subtrees, and combining them all at the end to get the probability of the gene tree according to the multispecies coalescent, given the current parameter values. Cutting the gene tree into species-specific subtrees is quite easy, because we can use the dates of speciation events to know what's before and after speciation events. The resulting subtrees are represented with the grey boxes in Fig. 2. In this figure, each subtree corresponds to one particular population, either extant or ancestral. Inside each subtree, given its length, the effective population size, and dates of coalescence (alleles splitting), the coalescent model provides simple formulas for computing the probability of the gene subtree given other parameters. These subtree probabilities are then multiplied to get the gene tree probability given current parameter values.

Two parameters associated to branches of the species tree have a direct impact on the expected amount of gene tree-species tree incongruence:

- **Time between speciations.** The more a branch length increases, the more the pool of alleles is expected to change. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are long.
- **Effective population size between speciations.** In populations with small effective population sizes, chance events can cause large shifts in allele frequencies, and possibly disappearance of alleles, irrespective of the fitness of this allele. In large populations, because an allele is likely carried by a large number of individuals, its disappearance is less likely, the population of alleles is more stable. Alleles are therefore less likely to persist for several speciation events if the branches between these speciation events are characterised by small effective population sizes.

Overall, larger amounts of gene tree-species tree incongruence are expected in phylogenies characterised by short branches with large population sizes. A corollary of that is that larger amounts of gene tree-gene tree incongruence are expected as well. To measure the susceptibility of species phylogenies to generate

incomplete lineage sorting, the concept of *coalescent time units* has been introduced. Coalescent time units are obtained when branch length λ is divided by effective population size N_e. As a consequence, in a species tree whose branches are expressed in coalescent time units, a branch length of 1 *coalescent time unit* means a branch length of N_e *generations*. Once branch lengths on the species tree are measured in coalescent time units, it becomes easy to spot species trees that generate a lot of incongruence: those are short trees.

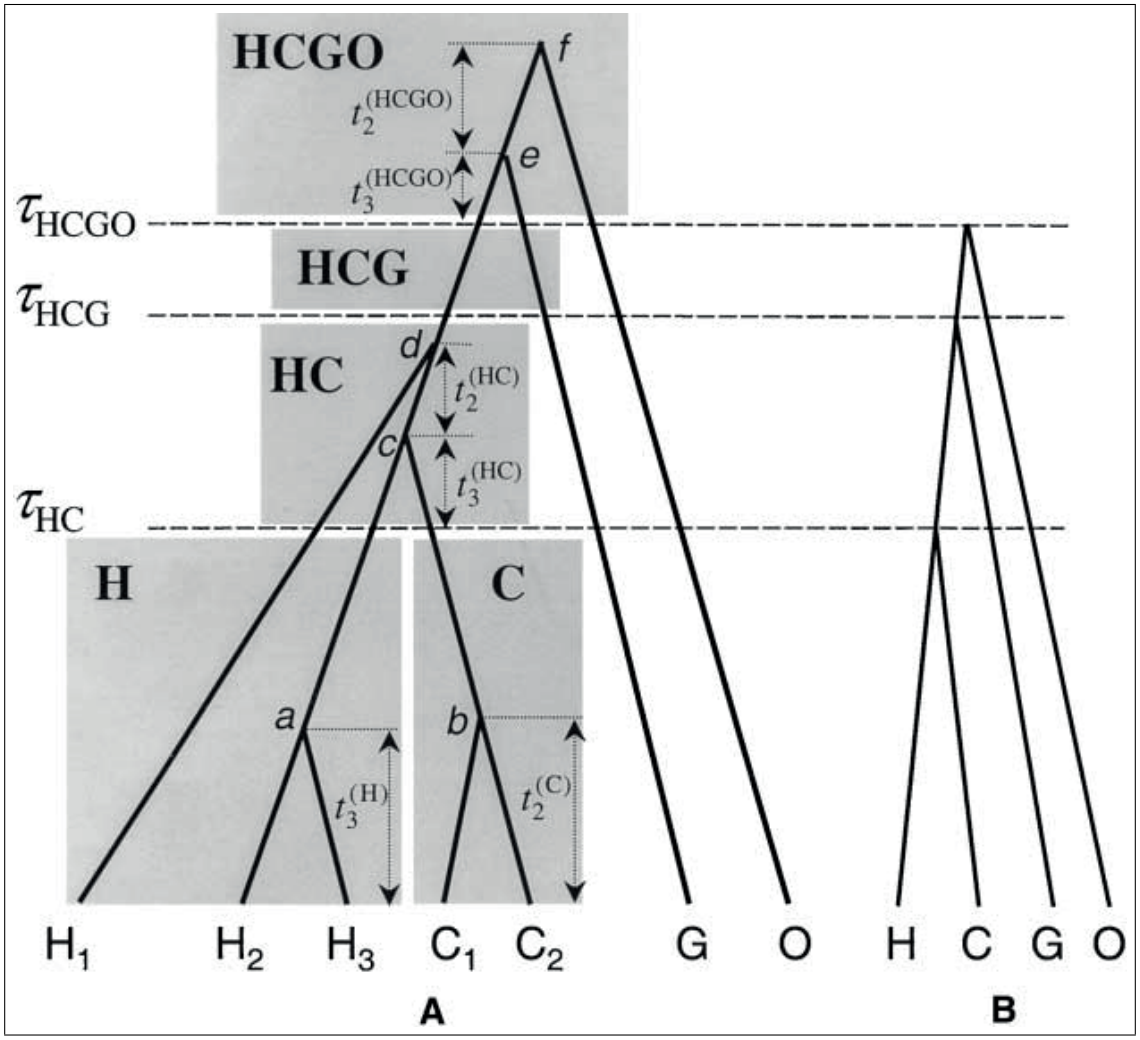

Figure 2: The multispecies coalescent. A) A gene tree, including 3 human alleles, 2 Chimp alleles, one Gorilla allele, and one Orang-outan allele. τ parameters are speciation times, t parameters are divergence time in the gene tree, the grey squares represent the ancestral populations, with their respeciive sizes. B) The corresponding species tree. In this model, the speciation times define minimal boundaries for allele divergence times. [Replicated from Fig. 1 in Rannala and Yang (2003).]

The exercises assume you have a working installation of RevBayes. The first exercise aims at understanding the impact of the parameters of the multispecies coalescent. The other exercises will introduce methods to reconstruct species trees and gene trees using the multispecies coalescent or closely-related models. Apart from the first exercise which requires copying and pasting from this document, scripts are all placed in

tutorials/RB_GeneTreeSpeciesTree_Tutorial/RB_tutorial_files/RevBayes_scripts/, in four folders. *It is highly recommended to start RevBayes from each of these folders when doing the exercises.*

First RevBayes exercise: simulating gene trees under the multispecies coalescent

1. Open RevBayes
2. Let's simulate a species tree with 10 taxa, 10 gene trees, 5 alleles per species (feel free to change these values).

```
n_species <- 10
n_genes <- 10
n_alleles <- 5
```

3. We simulate a species tree topology according to a birth-death process with arbitrary parameter values (similar to Leaché and Rannala (2011)):

```
speciation ~ exponential(10.0)
extinction ~ exponential(10.0)
tree_height ~ unif(0,1.0)
speciation.setValue(2)
extinction.setValue(0.3)
tree_height.setValue(0.8)
speciesTree ~ cBDP(lambda=speciation, mu=extinction, origin=
   tree_height, nTaxa=n_species, names=s_names)
```

4. Then we can use the multispecies coalescent model to generate gene trees. These can be examined using Figtree or NJplot or any other tree viewer, but we can also directly compute symmetric differences between these from RevBayes. First, we simulate a set of gene trees, using a single effective population size for all branches, and after having constructed a map between species names and gene names:

```
# Build the mapping between sequence names and species names.
for (i in 1:n_species) {
        for (j in 1:n_alleles) {
                taxa[(i-1)*n_alleles+j] <- taxon(taxonName=s_names[i
                   ]+"_"+j, speciesName=s_names[i])
        }
}
# Set the effective population size
Ne ~ gamma(shape=0.1,rate=0.1)
Ne.setValue(0.004)
# Simulate gene trees
for (i in 1:n_genes) {
   # The gene tree from the multispecies coalescent process
   # Note that if Ne had been a vector of effective population sizes
      instead of a single value,
   # allowing 1 parameter per branch of the species tree, the same
      line would work.
   geneTrees[i] ~ dnConstPopMultispCoal(speciesTree=speciesTree, Ne=
      Ne, taxa=taxa)
}
```

5. We can compute symmetric differences between all these gene trees. The symmetric difference between two trees is the total number of partitions found in one tree but not in the other one. In our case, the maximal difference is as follows:

```
maxDiff <- 2 * (n_species*n_alleles - 2)
```

6. That will give us a reference for comparing with the values we get on our gene trees. We can build a function for computing all pairwise symmetric differences between our gene trees, and getting the mean.

```
function RealPos symDiffVector ( Real[] vec ) {
        ndiff <- 1
        for (k in 1:(n_genes-1)) {
                for (j in (k+1):n_genes) {
                        diff[ndiff]<-symDiff (geneTrees[k], geneTrees
                           [j])
                        ndiff <- ndiff+1
                }
        }
        return (mean(diff))
}
#We can then use this function on our gene trees:
symDiffVector(geneTrees)
```

7. Now to get a sense of how population size and branch lengths alter the gene tree distribution, we can relaunch the multispecies coalescent simulation (step 4) and look at the resulting gene trees after having rescaled the species tree or changed the effective population size. To do these little changes:

```
# Changing Ne:
Ne.setValue(0.08)
#Rescaling the species tree:
speciesTree.rescale(0.1)
```

Using the functions above, it is possible to look at the species tree in coalescent time units (which is very convenient). How would you do that?

Do these observations seem coherent with the multispecies coalescent presentation above?

2 Alternatives to the multispecies coalescent model

2.1 Strengths and weaknesses of the multispecies coalescent

The multispecies coalescent model is an elegant model. As we have seen above, we can easily simulate data under this model. Inference using this model, combined with a model of sequence evolution, is also possible, and when it works, is very informative. Not only can we get a dated species tree and dated gene trees from the multispecies coalescent, we can also get extant and ancestral effective population sizes (Edwards, Liu and Pearl, 2007; Heled and Drummond, 2010). However inferring many parameters is difficult, and convergence can be difficult to reach with such models. In particular, the strong interdependence that exists between the gene trees and the species tree makes it easy for algorithms to fall into local maxima. As a consequence, there are ongoing efforts to develop methods for which inference would be easier, albeit at the cost of approximations and simplifications.

2.2 Alternatives to the multispecies coalescent

An easy way to simplify the problem is to consider that parts of it are already solved. For instance, several approaches assume that rooted gene trees are available. The problem then becomes markedly easier, but inference is then highly dependent on the quality of the input gene trees. Often, such methods also make other simplifying assumptions, and e.g. do not try to estimate separately time and effective population sizes, but instead directly work with coalescent time units. These methods usually are much faster than the multispecies coalescent, should be more robust against local maxima, but are less ambitious about the amount of information they can get from the data, and are can be sensitive to the quality of the input gene trees.

Another approach is to use methods that mathematically bypass estimating gene trees altogether. To our knowledge, there are three such approaches: SNAPP (Bryant et al., 2012), POMO (De Maio, Schlötterer and Kosiol, 2013), and SVDQuartets (Chifman and Kubatko, 2014). They differ in the way the algorithms work, but they are all based on the same idea, which is integrating out gene trees. To achieve that they extend the model of sequence evolution, which usually models substitution events, to also model population-level processes. In RevBayes, so far only the POMO model has been implemented and will be discussed in the following part.

2.3 The POMO model

POMO models allele frequency changes along with mutations with a single transition matrix. It extends the usual 4×4 DNA substitution matrix to incorporate polymorphic states. In doing so, it makes a first important approximation: it only considers biallelic states. For instance, it considers sites at which either an A or a C is found in a population, but it won't consider sites at which 3 different states, e.g. A, C, T, are observed in a population. Then it makes a second approximation, which introduces a single virtual population size in lieu of the branchwise effective population sizes. This virtual population size is not inferred, but is fixed to some low number. In practice, De Maio, Schlötterer and Kosiol (2013) consider that a virtual population size of 10 individuals should produce good results. This virtual population size directly constrains the types of polymorphic states that can be considered. With 10 individuals for instance, only frequencies such as $(100\%A)$, $(10\%A, 90\%C)$, $(20\%A, 80\%C)$, $(30\%A, 70\%C)$, ..., $(90\%A, 10\%C)$, $(100\%C)$ can be considered. The POMO matrix models transitions among all these states, polymorphic ones as well as monomorphic ones, and has a size that depends on the virtual population size. For instance, with a virtual population size of 10 individuals, the POMO square matrix contains 58 rows: 4 monomorphic states, plus 6 types of biallelic states (AC, AG, AT, CG, CT, GT) times 9 frequencies. Additional assumptions of the POMO model include total independence among sites (no linkage among sites), and absence of mutations in biallelic states: transitions among biallelic states or from biallelic states to monoallelic states only occur through population-level changes in allele frequencies, not through mutation of one allele into another. Mutations only occur to transit from a monoallelic state to a biallelic state.

The POMO model therefore makes several approximations to avoid estimating gene trees. Fewer parameters need to be estimated, as neither gene trees nor population sizes are estimated, but other parameters can be introduced into the model. For instance De Maio, Schlötterer and Kosiol (2013) estimate 4 base fitness parameters, which they use to model GC-biased gene conversion, which tends to increase the GC content of recombining sequences. In the RevBayes implementation of POMO, base fitnesses can be estimated as well.

3 Inference using the multispecies coalescent, the POMO model, and basic concatenation

In this section we will perform inference using the multispecies coalescent, the POMO model, and concatenation. Depending on your machine and on the size of the data, successful inference may take some time. If this tutorial is done in a classroom environment, it may be wise to convene with friends that one tries the multispecies coalescent model while another one tries the POMO model, and the third one tries the concatenation approach, and that results will be shared. First we will simulate data, and then we will perform inference using the three different approaches.

3.1 Simulating data

We are going to do inference on simulated data, as produced during the previous section. However in addition to the species tree and the gene trees, we will be producing sequence data. Below is the RevBayes code necessary to simulate all the data. As before, parameters of the simulation can be changed as seems fit, but here we chose parameters that resemble Leaché and Rannala (2011).

WARNING: The following script may be less up to date than the simulation scripts named "MultiSpeciesCoalescentSimulatingTreesAndAlignments.Rev".

First we can set the folder in which we will save the output of our work.

```
dataFolder<-"/PATH/TO/A/FOLDER/WHERE/TO/SAVE/THE/OUTPUT"
setwd(dataFolder)
```

Let's simulate a species tree with 6 taxa, 10 gene trees, 5 alleles per species, and along each gene tree one gene alignment 200 bases long. It's a small data set, designed for manageable inference during a tutorial, but patient users may want to simulate a larger data set.

```
n_species <- 10
n_genes <- 10
n_alleles <- 5
n_sites <- 100
n_branches <- 2 * n_species - 3 # number of branches in a rooted tree
```

The species-tree birth-death model:

```
# Species names
for (i in 1:n_species) {
        s_names[i] <- "Species_"+i
}
speciation ~ exponential(10.0)
extinction ~ exponential(10.0)
tree_height ~ unif(0,1.0)
speciation.setValue(2)
extinction.setValue(0.3)
tree_height.setValue(0.008)
# Species tree from birth-death process
speciesTree ~ cBDP(lambda=speciation, mu=extinction, origin=tree_height,
   nTaxa=n_species, names=s_names)
# Making a backup for future reference:
trueSpeciesTree <- speciesTree
```

The gene-tree multispecies coalescent model

```
# Build the mapping between sequence names and species names.
for (i in 1:n_species) {
        for (j in 1:n_alleles) {
                taxa[(i-1)*n_alleles+j] <- taxon(taxonName=s_names[i]+"_
                   "+j, speciesName=s_names[i])
        }
}
# Set the effective population size
Ne ~ gamma(shape=0.5,rate=0.5)
Ne.setValue(0.004)
# Simulate gene trees
for (i in 1:n_genes) {
   # The gene tree from the multispecies coalescent process
   # Note that if Ne had been a vector of effective population sizes,
   # allowing 1 parameter per branch of the species tree, the same line
      would work.
   geneTrees[i] ~ dnConstPopMultispCoal(speciesTree=speciesTree, Ne=Ne,
      taxa=taxa)
}
# Making a backup for future reference:
trueGeneTrees <- geneTrees
trueNe <- Ne
```

Substitution models will all be based on the GTR model. However, the parameters of the models change from one gene family to the next.

```
for (i in 1:n_genes) {
  er_prior[i] <- v(1,1,1,1,1,1)
  er[i] ~ dnDirichlet(er_prior[i])
  sf_prior[i] <- v(1,1,1,1)
  sf[i] ~ dnDirichlet(sf_prior[i])
  Q[i] := gtr(er[i],sf[i])
}
# Making a backup for future reference:
for (i in 1:n_genes) {
        trueEr[i] <- er[i]
        trueSf[i] <- sf[i]
}
```

Then we assume a strict clock Model, but of course we could assume a relaxed clock. Each gene family can have its own rate of evolution, drawn from an exponential distribution.

```
for (i in 1:n_genes) {
  familyRates[i] ~ exponential(1.0)
}
# Making a backup for future reference:
for (i in 1:n_genes) {
  trueFamilyRates[i] <- familyRates[i]
}
```

We add a model of among-site rate variation, handled by a discretized Gamma distribution, one for each gene family:

```
for (i in 1:n_genes) {
  shape_prior[i] <- 0.05
  shape[i] ~ dnExponential( shape_prior[i] )
  norm_gamma_rates[i] := discretizeGamma( shape[i], shape[i], 4, false )
}
```

Finally we link it all using the PhyloCTMC model, which simulates gene alignments:

```
for (i in 1:n_genes) {
  alns[i] ~ phyloCTMC(tree=geneTrees[i], Q=Q[i],  branchRates=familyRates
     [i], siteRates=norm_gamma_rates[i], nSites=n_sites, type="DNA")
}
```

We can then save the simulated data to the folder chosen at the beginning of this script. We need to save the species tree, the gene trees, and the gene alignments

```
write(speciesTree, filename="speciesTree")
for (i in 1:n_genes) {
        write(geneTrees[i], filename="geneTree_"+i)
}
for (i in 1:n_genes) {
        writeFasta(alns[i], filename="alignment_"+i+".fasta")
}
```

3.2 Inference using the multispecies coalescent

WARNING: The following script may be less up to date than the simulation script named "MultiSpeciesCoalescentWithSequences/MultiSpeciesCoalescentWithSequencesMCMC.Rev". In the following we perform inference from the alignments, but we could also do inference directly from the gene trees, as can be seen in the script "MultiSpeciesCoalescent/MultiSpeciesCoalescentMCMC.Rev".

Now that we have simulated data, we can run inference under the multispecies coalescent, using the sequences as input data.

First we clamp the alignments:

```
for (i in 1:n_genes) {
        alns[i].clamp(alns[i])
}
```

Then we change the starting values, because we want to start from random values, not the values used in the simulation.

```
# Redrawing the parameters of the birth-death process:
speciation.redraw()
extinction.redraw()
tree_height.redraw()
# Redrawing the species tree:
speciesTree.redraw()
# Redrawing the parameter Ne:
Ne.redraw()
# Redrawing the gene trees:
for (i in 1:n_genes) {
   geneTrees[i].redraw()
}
# Redrawing the parameters of the substitution models:
for (i in 1:n_genes) {
  er[i].redraw()
  sf[i].redraw()
}
# Idem for the family-wise rates of sequence evolution:
for (i in 1:n_genes) {
  familyRates[i].redraw()
}
# Idem for the family-wise across-site rate variation parameters:
for (i in 1:n_genes) {
  shape[i].redraw()
}
```

We need to set up moves for the birth-death parameters, the species tree topology, the gene tree topologies, the parameter Ne, the parameters of the substitution models, the rates on the gene trees.

```
moveIndex <- 0
# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,1,true,1.0) # In the revLanguage
   , table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)
# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 1.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 1.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, delta=1.0, tune=true,
   weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)
moves[moveIndex++] <- mvRootTimeSlide(speciesTree, 1.0, true, 3.0)
# moves on the gene tree topologies and node ages
for (i in 1:n_genes) {
   moves[moveIndex++] <- mvNNI(geneTrees[i], 1.0)
   moves[moveIndex++] <- mvFNPR(geneTrees[i], 1.0)
   moves[moveIndex++] <- mvSubtreeScale(geneTrees[i], 5.0)
   moves[moveIndex++] <- mvTreeScale(geneTrees[i], delta=1.0, tune=true,
      weight=3.0)
   moves[moveIndex++] <- mvNodeTimeSlideUniform(geneTrees[i], 10.0)
   moves[moveIndex++] <- mvRootTimeSlide(geneTrees[i], 1.0, true, 3.0)
}
# move on Ne, the effective population size
moves[moveIndex++] <- mvScale(Ne,1,true,1.0)
# moves on the parameters of the substitution models
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvSimplexElementScale(er[i], alpha=10, tune=true,
      weight=3)
  moves[moveIndex++] <- mvSimplexElementScale(sf[i], alpha=10, tune=true,
      weight=2)
}
# moves on the family-wise rates
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvScale(familyRates[i], lambda=0.8, tune=true,
     weight=3.0)
}
# moves on the across-sites rate variation parameters:
for (i in 1:n_genes) {
  moves[moveIndex++] <- mvScale(shape[i], lambda=0.8, tune=true, weight
     =3.0)
}
```

Then we define a few monitors to keep track of how things go. First, basic monitors:

```
mntrIndex <- 0
# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- modelmonitor(filename="
   multispeciesCoalescent_clock.log",printgen=10, separator = "    ")
# One monitor to print the species trees sampled in the course of the
   MCMC:
monitors[mntrIndex++] <- filemonitor(filename="
   multispeciesCoalescent_clock_species.trees",printgen=10, separator = "
      ", speciesTree)
# One monitor for each gene family tree:
for (i in 1:n_genes) {
   monitors[mntrIndex++] <- filemonitor(filename="
      multispeciesCoalescent_clock_gene_"+ i +".trees",printgen=10,
      separator = "   ", geneTrees[i])
}
```

We will also compare our reconstructed parameters (trees or other variables) to the true values that were used in the simulation. First, examining the species tree and the gene trees.

```
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)
for (i in 1:n_genes) {
        distGeneTree[i] := symDiff (trueGeneTrees[i], geneTrees[i])
}
# We get one average value for all gene trees:
meanDistGeneTree := mean(distGeneTree)
```

We can also look at the values of some other variables:

```
# For Ne:
distNe := Ne - trueNe
# For equilibrium values of the GTR matrices, we want one index of how
   far we are.
# To achieve this we need to write some functions:
clear(i)
distEqFreq := diffVectorsOfVectors(trueSf, sf)
function RealPos diffVectors ( Real[] xvec, Real[] yvec ) {
        DI <- 0.0
        for (i in 1:xvec.size()) {
                DI <- DI + (xvec[i] - yvec[i])*(xvec[i] - yvec[i])
        }
        return DI
}
function RealPos diffVectorsOfVectors ( Real[][] xvecvec, Real[][]
   yvecvec ) {
        DA <- 0.0
        for (i in 1:xvecvec.size()) {
                DA <- DA + diffVectors(xvecvec[i], yvecvec[i])
        }
        return DA
}
function RealPos diffVectorsOfVectors ( Simplex[] xvecvec, Simplex[]
   yvecvec ) {
        DA <- 0.0
        for (i in 1:xvecvec.size()) {
                DA <- DA + diffVectors(xvecvec[i], yvecvec[i])
        }
        return xvecvec[2]
}
# Same thing for exchangeability parameters:
distExchange := diffVectorsOfVectors(trueEr, er)
monitors[mntrIndex++] <- screenmonitor(printgen=10, distExchange)
# Same thing for the family-wise rates
distRates := diffVectors(trueFamilyRates, familyRates)
monitors[mntrIndex++] <- screenmonitor(printgen=10, distRates)
# We can use one monitor that will output on the screen one parameter, Ne
   , distNe, distSpeciesTree, meanDistGeneTree, and distEqFreq:
monitors[mntrIndex++] <- screenmonitor(printgen=10, Ne, distNe,
   distSpeciesTree, meanDistGeneTree, distEqFreq)
# We could do similar things for the few remaining parameters, but really
    I think that's enough.
```

We then define our model and the mcmc object. We can use any node of our model as a handle, here we choose to use the species tree.

```
mymodel <- model(speciesTree)
# We create the MCMC object
mymcmc <- mcmc(mymodel, monitors, moves)
```

We finally launch the analysis.

```
# If we want to specify the amount of burnin:
# mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=400)
```

After the analysis, we analyze the output. We analyze the tree trace as saved in one of the output files.

```
# Let us start by reading in the tree trace
 treetrace <- readTreeTrace("multispeciesCoalescent_clock_species.trees")
# and get the summary of the tree trace
 treetrace.summarize()

# We output the Maximum A Posteriori tree
 mapTree(treetrace,"primates_clock_MAP.tre")
```

Of course, we could do the same analysis with each of the gene trees, possibly using a "for" loop in the rev language.

3.3 Inference using the POMO model

WARNING: The following script may be less up to date than the simulation script named "MultiSpeciesCoalescentAndPomo/MultiSpeciesCoalescentPomoMCMC.Rev".

We run the POMO model on the same data set simulated under the multispecies coalescent.

First, we need to prepare the data for analysis using POMO: concatenating the alignments, then transforming the state space from 4 bases to the POMO state space. We arbitrarily decide to use a virtual population size of size 10, as suggested in De Maio, Schlötterer and Kosiol (2013).

```
# First, concatenating the alignments:
concatenate<-alns[1]
for (i in 2:n_genes) {
        concatenate <- concatenate + alns[i]
}

virtual_population_size <- 10

# Now, we need to convert the DNA alignment into an alignment in the
   correct POMO state space.

pomo_concatenate <- pomoStateConvert(concatenate, virtual_population_size
   , taxa)
concatenate_n_sites <- pomo_concatenate.nchar()[1]
```

Now we have the data for doing inference using the POMO model. To define a POMO model, one needs several components. First, one needs to define a transition matrix for DNA mutations. Here we are going to use a GTR matrix.

```
er_prior <- v(1,1,1,1,1,1)
er ~ dnDirichlet(er_prior)

sf_prior <- v(1,1,1,1)
sf ~ dnDirichlet(sf_prior)

Q := gtr(er,sf)
```

Second, one can have different fitnesses for A, C, G, T. Here, we are going to assume that all 4 bases have the same fitness.

```
base_fitnesses <- v(1, 1, 1, 1)
```

Now we have all the elements to build a POMO matrix to model the evolution of a population of alleles along a species tree.

```
P := pomo(Q, base_fitnesses, virtual_population_size)
```

We also need to define root frequencies for all the states. To do that we need two variables:

```
# First, the proportion of polymorphic sites at the root
root_polymorphism_proportion ~ beta(alpha=1,beta=1)

# Second, one needs the root frequencies (we could use those of the GTR
   matrix but choose to have a non-stationary model):
root_base_frequencies ~ dnDirichlet(sf_prior)
```

Now we have all the elements to construct root frequencies for all the states:

```
root_frequencies := pomoRF (root_base_frequencies,
   root_polymorphism_proportion, Q, virtual_population_size)
#simplex_root_frequencies := simplex(root_frequencies[1],
   root_frequencies[2], root_frequencies[3], root_frequencies[4])
```

Adding an across site rate variation model, the usual Gamma distribution discretized in 4 categories:

```
shape_prior <- 0.05
shape ~ dnExponential( shape_prior )
norm_gamma_rates := discretizeGamma( shape, shape, 4, false )

# We do not assume variation among branches in rates
branch_rates <- 1.0

# We do not assume a proportion of invariant
p_inv ~ beta(alpha=1,beta=1)
p_inv.setValue(0.000001)
```

Combining it all with the PhyloCTMC Model and clamp the model to the data:

```
aln ~ phyloCTMC(tree=speciesTree, Q=P, rootFreq=root_frequencies,
   branchRates=branch_rates, siteRates=norm_gamma_rates, pInv=p_inv,
   nSites=concatenate_n_sites, type="Standard")

aln.clamp(pomo_concatenate)
```

Changing the starting values, we want to start from random values, not the values used in the simulation:

```
# Redrawing the parameters of the birth-death process:
speciation.redraw()
extinction.redraw()
tree_height.redraw()

# Redrawing the species tree:
speciesTree.redraw()
```

Then we need to set up moves for the birth-death parameters, the species tree topology, the gene tree topologies, the parameter Ne, the parameters of the substitution models, the rates on the gene trees.

```
moveIndex <- 0

# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,1,true,1.0) # In the revLanguage
   , table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)


# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 1.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 1.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, delta=1.0, tune=true,
   weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)
moves[moveIndex++] <- mvRootTimeSlide(speciesTree, 1.0, true, 3.0)

# moves on the parameters of the substitution model
moves[moveIndex++] <- mvSimplexElementScale(er, alpha=10, tune=true,
   weight=3)
moves[moveIndex++] <- mvSimplexElementScale(sf, alpha=10, tune=true,
   weight=2)

# moves on the 4 fitness values
# SH: These are not stochastic nodes, so moves cannot operate on them!
#moves[moveIndex++] <- mvScale(base_fitnesses[1],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[2],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[3],1,true,1.0)
#moves[moveIndex++] <- mvScale(base_fitnesses[4],1,true,1.0)

# moves on the parameters of the root frequencies
moves[moveIndex++] <- mvSimplexElementScale(root_base_frequencies, alpha
   =10, tune=true, weight=3)
#moves[moveIndex++] <- mvScale(root_polymorphism_proportion, lambda=10,
   tune=true, weight=2)
moves[moveIndex++] <- mvSlide(root_polymorphism_proportion, delta=10,
   tune=true, weight=2)

# moves on the across-sites rate variation parameter:
moves[moveIndex++] <- mvScale(shape, lambda=0.8, tune=true, weight=3.0)
```

Then we define a few monitors to keep track of how things go:

```
mntrIndex <- 0

# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- modelmonitor(filename="pomo_clock.log",printgen
   =10, separator = "      ")

# One monitor to print the species trees sampled in the course of the
   MCMC:
monitors[mntrIndex++] <- filemonitor(filename="pomo_clock_species.trees",
   printgen=10, separator = "     ", speciesTree)

# We also want to monitor how far we are from the true values, which we
   have because we rely on simulations.
# First, we can compute the distance between the reconstructed and the
   true species tree:
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)


# We can use one monitor that will output on the screen one parameter, Ne
   , distNe, distSpeciesTree, meanDistGeneTree, and distEqFreq:
monitors[mntrIndex++] <- screenmonitor(printgen=10, distSpeciesTree)
```

We define our model:

```
# First we need to get rid of several variables that are not part of the
   Pomo model.
clear(alns)
clear(geneTrees)
clear(Ne)
clear(familyRates)
clear(trueEr)
clear(trueFamilyRates)
clear(trueGeneTrees)
clear(trueNe)
clear(trueSf)

# We can use any node of our model as a handle, here we choose to use the
    species tree.

mymodel <- model(speciesTree)
```

We create and run the MCMC object

```
mymcmc <- mcmc(mymodel, monitors, moves)
# mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=40000)
```

3.4 Inference using basic concatenation

WARNING: The following script may be less up to date than the simulation scripts named "MultiSpeciesCoalescentAndConcatenation/MultiSpeciesCoalescentConcatenationMCMC.Rev".

We can also concatenate all alignments and run a partitioned GTR model on it.

```
# Birth-Death process priors.
speciation ~ exponential(10.0)
extinction ~ exponential(10.0)
tree_height ~ unif(0.0,100.0)
speciation.setValue(2)
extinction.setValue(0.3)

# Species tree from birth-death process
#We assume 1000,000 generations
speciesTree ~ cBDP(lambda=speciation, mu=extinction, rootAge=tree_height,
    nTaxa=n_species, names=s_names)
```

Second, one needs to define a transition matrix for DNA mutations, and we use a GTR matrix for that.

```
er_prior <- v(1,1,1,1,1,1)
er ~ dnDirichlet(er_prior)

sf_prior <- v(1,1,1,1)
sf ~ dnDirichlet(sf_prior)

Q := gtr(er,sf)
```

Adding an across site rate variation model, the usual Gamma distribution discretized in 4 categories.

```
shape_prior <- 0.05
shape ~ dnExponential( shape_prior )
norm_gamma_rates := discretizeGamma( shape, shape, 4, false )

# We do not assume variation among branches in rates
branch_rates <- 1.0

# We do not assume a proportion of invariant
p_inv ~ beta(alpha=1,beta=1)
p_inv.setValue(0.000001)
```

To link all the parts of the model together we use the phyloCTMC object and clamp it to the concatenate.

```
aln ~ phyloCTMC(tree=speciesTree, Q=Q, branchRates=branch_rates,
   siteRates=norm_gamma_rates, pInv=p_inv, type="DNA")

aln.clamp( concatenate )
```

Then we define the moves.

```
moveIndex <- 0

# moves for the birth-death parameters
moves[moveIndex++] <- mvScale(speciation,lambda=1,tune=true,weight=1.0) #
    In the revLanguage, table indices start at 1
moves[moveIndex++] <- mvScale(extinction,1,true,1.0)
moves[moveIndex++] <- mvSlide(tree_height,delta=1.0,true,2.0)

# moves on the tree topology and node ages
moves[moveIndex++] <- mvNNI(speciesTree, 2.0)
moves[moveIndex++] <- mvNarrow(speciesTree, 5.0)
moves[moveIndex++] <- mvFNPR(speciesTree, 2.0)
moves[moveIndex++] <- mvSubtreeScale(speciesTree, 5.0)
moves[moveIndex++] <- mvTreeScale(speciesTree, tree_height, delta=1.0,
   tune=true, weight=3.0)
moves[moveIndex++] <- mvNodeTimeSlideUniform(speciesTree, 10.0)

# moves on the parameters of the substitution model
moves[moveIndex++] <- mvSimplexElementScale(er, alpha=10, tune=true,
   weight=3)
moves[moveIndex++] <- mvSimplexElementScale(sf, alpha=10, tune=true,
   weight=2)

# moves on the across-sites rate variation parameter:
moves[moveIndex++] <- mvScale(shape, lambda=0.8, tune=true, weight=3.0)
```

We define a few monitors to keep track of how things go.

```
mntrIndex <- 0

# One monitor to backup the parameters, in case we want to stop and
   restart the analysis:
monitors[mntrIndex++] <- modelmonitor(filename=dataFolder+"
   concatenation_clock.log",printgen=10, separator = "  ")

# One monitor to print the species trees sampled in the course of the
   MCMC:
monitors[mntrIndex++] <- filemonitor(filename=dataFolder+"
   concatenation_clock_species.trees",printgen=10, separator = " ",
   speciesTree)

# We also want to monitor how far we are from the true values, which we
   have because we rely on simulations.
# First, we can compute the distance between the reconstructed and the
   true species tree:
distSpeciesTree := symDiff (trueSpeciesTree, speciesTree)


# We can use one monitor that will output on the screen one parameter, Ne
   , distNe, distSpeciesTree:
monitors[mntrIndex++] <- screenmonitor(printgen=10, distSpeciesTree)
```

We define the model and launch the MCMC.

```
mymodel <- model(speciesTree)

# We create the MCMC object
mymcmc <- mcmc(mymodel, monitors, moves)

#mymcmc.burnin(generations=200,tuningInterval=100)
mymcmc.run(generations=4000)
```

Batch Mode

If you wish to run this exercise in batch mode, the files are provided for you.

You can carry out these batch commands by providing the file name when you execute the **rb** binary in your unix terminal (this will overwrite all of your existing run files).

- `$ rb NameOfTheFile.Rev`

Useful Links

- RevBayes: https://github.com/revbayes/code
- Tree Thinkers: http://treethinkers.org

Questions about this tutorial can be directed to:

- Bastien Boussau (email: boussau@gmail.com)
- Sebastian Höhna (email: sebastian.hoehna@gmail.com)

Version dated: August 28, 2014

Relevant References

Boussau B. 2009. Phylogenetic Relationships Deduced from Whole Genome Comparisons. .

Bryant D, Bouckaert R, Felsenstein J, Rosenberg Na, RoyChoudhury A. 2012. Inferring species trees directly from biallelic genetic markers: bypassing gene trees in a full coalescent analysis. Molecular biology and evolution. 29:1917–32.

Chifman J, Kubatko L. 2014. Quartet Inference from SNP Data Under the Coalescent Model. Bioinformatics (Oxford, England). .

De Maio N, Schlötterer C, Kosiol C. 2013. Linking Great Apes Genome Evolution across Time Scales Using Polymorphism-Aware Phylogenetic Models. Molecular biology and evolution. 30:2249–62.

Edwards SV, Liu L, Pearl DK. 2007. High-resolution species trees without concatenation. Proceedings of the National Academy of Sciences of the United States of America. 104:5936–5941.

Heled J, Drummond AJ. 2010. Bayesian Inference of Species Trees from Multilocus Data. Molecular Biology and Evolution. 27:570.

Leaché AD, Rannala B. 2011. The accuracy of species tree estimation under simulation: a comparison of methods. Systematic Biology. 60:126–137.

Rannala B, Yang Z. 2003. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. Genetics. 164:1645–1656.

Szöllősi GJ, Tannier E, Daubin V, Boussau B. 2014. The inference of gene trees with species trees. Systematic Biology. .

Zuckerkandl E, Pauling L. 1965. Evolutionary divergence and convergence in proteins BT - Evolving genes and proteins. In: Bryson V, Vogel HJ, editors, Evolving genes and proteins, New York: Academic Press, pp. 97–166.