Dating phylogenetic trees with McmcDate

A detailed tutorial

Dominik Schrempf

February 28, 2023

Abstract

McmcDate can date a phylogenetic tree with constraints. That is, it can estimate the ages of ancestral nodes of phylogenetic trees with node age calibrations, node order constraints, and node braces. McmcDate is fast because it approximates the phylogenetic likelihood with a multivariate normal distribution.

Here, I briefly review the process of dating phylogenetic trees and provide an example dating analysis of a tree of eukaryotes. I use data from Strassert et al. (2021) who provide the alignment and node age calibrations from fossils. Moreover, I use node order constraints from two possible successions of plastid endosymbiosis events.

Section 4 describes the example analysis. If you want to follow the tutorial on your computer, install the Haskell toolchain and clone the McmcDate repository as described in Appendix C.

Contents

1 Definitions

We use the term *(rooted, bifurcating) tree* to denote a directed acyclic graph with node labels and branch lengths in which (i) not more than one branch connects any two nodes, and (ii) all nodes have degree three or less. Usually, nodes correspond to (ancestral) species, and branch lengths to sequence distance or time. We use the term *(rooted, bifurcating) topology* to denote a (rooted, bifurcating) tree without information about branch lengths. We put the words *rooted* and *bifurcating* into parentheses, because they are sometimes omitted. The root of the tree or topology is the only node with all branches pointing away to other nodes. The root is the oldest node. Leaves are extant nodes with a single branch pointing towards them.

For a given topology, *dating the tree*[1] corresponds to finding branch lengths for this topology which are measured in absolute time units (for example, in Million years), and which describe the data in the *best* way. *Best* can mean different things, and we will carefully analyze what *best* means in our case. Sometimes, all we can do is date a tree with relative time units (Section 2.1).

Similarly, we use the term *unrooted (bifurcating) tree* to denote an undirected acyclic graph with node labels and branch lengths in which (i) not more than one branch connects any two nodes, and (ii) all nodes have degree three or less. Finally, we use the term *unrooted (bifurcating) topology* to denote an unrooted (bifurcating) tree without information about branch lengths. Again, we usually omit the word *bifurcating*. We need unrooted trees and topologies because we will use reversible substitution models (for a review, see Yang 2006) to infer unrooted trees with branch lengths measured in expected number of substitutions per site. Reversible substitution models are unable to discriminate between the two directions in time.

2 Dating a tree

Dating a tree is difficult because we want to estimate branch lengths measured in absolute time units. However, an alignment only contains information about distances measured in expected number of substitutions per site. In particular, it does not contain information about the evolutionary rates nor about the actual elapsed absolute time units. For a given branch,

$$\begin{aligned} &d\ [\text{expected number of substitutions}] = \\ &\quad r\ [\text{expected number of substitutions per year}] \cdot t\ [\text{years}], \end{aligned} \tag{1}$$

where d is the branch length measured in expected number of substitutions per site, r is the evolutionary rate on this branch, and t is the branch length

[1] We should probably say *dating the topology* but this phrase is not used.

measured in absolute time units. The situation is severe because r and t are confounded. If we multiply t with a positive number c and divide r by the same positive number c, the distance d stays constant. Since the likelihood only depends on the distance d we can not discriminate between parameter tuples (t, r) and (t', r'), where

$$t' = t \cdot c, \tag{2}$$

$$r' = r/c, \tag{3}$$

for *any* $c \in (0, \infty)$ assuming $t, r \in \mathbb{R}_+$. This means we need auxiliary data to constrain the ages of at least some nodes. There are three forms of constraints from auxiliary data which are explained in the next sections: (a) node age calibrations, (b) node order constraints, and (c) node braces. Appendix B details the specifications about how auxiliary data is used with McmcDate.

2.1 Node age calibrations

We call constraints on the ages of internal nodes *node age calibrations* (Yang and Rannala 2005). Without node age calibrations, all dated trees are equally likely! Since most phylogenetic dating methods are Bayesian, and impose prior functions on all of their parameters, the values of the posterior function of different trees still differ in a deceiving way. However, in the absence of node age calibrations, the differences in the values of the posterior function are exclusively caused by the chosen prior functions, and not by the information we have obtained from the data.

Even more, in order to limit the maximum ages of the nodes, we require at least one node age calibration with a maximum boundary. The closer in terms of absolute time units the calibrated node with a maximum boundary is to the root, the better. Ideally, we have a maximum age boundary for the root itself, but sometimes the root age boundaries are unknown. Section 4.3 provides details on how to add node age calibrations to your dating analysis with McmcDate.

I can not repeat this often enough: **Without proper node age calibrations**, and in particular, **without at least one maximum age boundary**, **dating a tree in absolute time units is impossible**. Without a single maximum node age boundary, we can still date the tree using branch lengths measured in *relative time units*. In particular, we achieve this by fixing the tree height to be 1.0. McmcDate automatically falls back to dating in relative time when no maximum node age boundary is found. If other programs date trees in absolute time units without a maximum age boundary, do not trust the results!

2.2 Node order constraints

Next to calibrating the absolute ages of nodes, we can also constrain the relative order of nodes on a tree (Szöllõsi et al. 2022). For example, due to external analyses, we may have detailed knowledge about a horizontal gene transfer. That is, we do know the exact donor and the recipient branches. In this case, a direct horizontal gene transfer can only happen, if these branches coexist for a period of time (Figure 1). If the branches do not coexist for a period of time but the recipient branch is younger than the donor branch, the horizontal gene

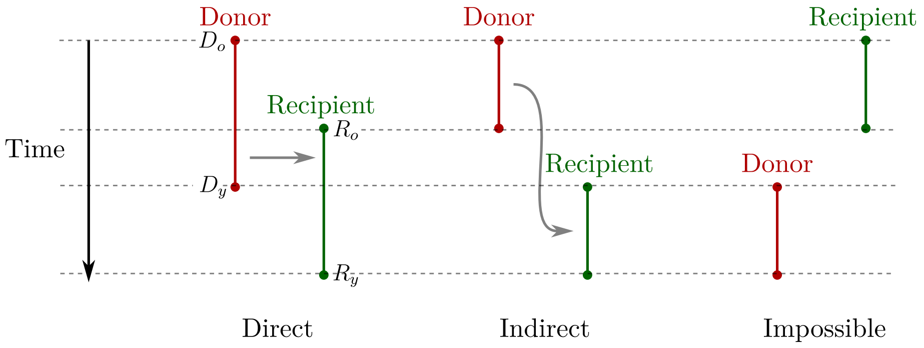

Figure 1: Direct, indirect and impossible horizontal gene transfers. A direct horizontal gene transfer happens between branches coexisting in time. An indirect horizontal gene transfer involves multiple events and an intermediate lineage. If the recipient branch is older than the donor branch, a horizontal gene transfer is impossible. For the direct transfer, D_o, D_y, R_o, and R_y denote the old and young ages of the delimiting nodes of the donor and recipient branches, respectively.

transfer can still happen indirectly through multiple events. An indirect gene transfer requires an intermediate lineage carrying the gene. The intermediate lineage has either gone extinct (Szöllősi et al. 2013), has not been sampled, or has lost the gene. Gene transfers are impossible if the recipient branch is older than the donor branch.

For example, let the ages of the old and young delimiting nodes of the donor and recipient branches be D_o, D_y, R_o, and R_y, respectively (Figure 1). Then, a direct horizontal gene transfer provides us with two node order constraints,

$$D_y < R_o, \text{ and } D_o > R_y, \tag{4}$$

where $<$ means *younger than* and $>$ means *older than.* If we allow indirect gene transfers, we only get one node order constraint,

$$D_o > R_y, \tag{5}$$

which is enough to prohibit impossible gene transfers (Figure 1).

Sometimes, we do not know the exact donor branch but only that the donor branch must be part of a known subtree. However, we usually do know the recipient branch which is the stem of the subtree comprising the species which contain the horizontally transferred gene. In this case, the ancestral node of the stem of the donor subtree has to be older than the descending node of the recipient branch.

2.3 Node braces

Finally, sometimes we know that two or more nodes have the same age or similar ages. For example, when analyzing gene trees, an early gene duplication event may separate the gene tree into two subtrees. Subsequent speciation events may be observable on both subtrees, and the corresponding speciation nodes should have similar ages. In this tutorial we will not brace nodes but see Appendix B.3 for the specifications.

3 Pipeline

McmcDate is fast because it uses a clever way to approximate the phylogenetic likelihood similar to MCMCTree (Yang and Rannala 2005). MCMCTree approximates the surface of the phylogenetic likelihood using Taylor expansion (Reis and Yang 2011). That is, a prior analysis computes the maximum likelihood together with the gradient and the Hessian matrix which MCMCTree uses in a subsequent analysis to date the tree.

Here, we use a similar technique, albeit with a Bayesian method to estimate the posterior distribution of branch lengths measured in expected number substitutions per site (Appendix A). In particular, the pipeline for dating a tree with McmcDate is:

1. Prepare a multi sequence alignment and an unrooted topology.
2. For this alignment and unrooted topology, sample from the distribution of unrooted trees with branch lengths measured in expected number of substitutions per site using any Bayesian approach.
3. Specify a rooted topology to date. Prepare auxiliary data such as node age calibrations or node order constraints.
4. Date the topology with McmcDate. In particular, sample from the distribution of rooted trees with branch lengths measured in absolute time units. If the node age calibrations are insufficient, we can still use relative time units.

Steps 3 and 4 are fast, especially when compared to Step 2. Hence, we can conveniently amend parameters used in the actual dating analysis, or compute dated trees for different roots. The following section shows an example analysis and describes the steps of the pipeline in detail.

4 Example analysis

4.1 Step 1: Alignment and unrooted topology

We are going to date a topology of eukaryotes (Strassert et al. 2021). The full published data set contains three alignments:

1. The alignment with the highest number of taxa comprises 733 eukaryotes with 62723 amino acids.
2. The authors used the first alignment to infer a tree with IQ-TREE (Minh et al. 2020) so they can filter taxa in an informed way (for example, to keep slow evolving taxa). The reduced data set comprises 136 operational taxonomic units with 73460 amino acids.
3. A small data set with 63 operational taxonomic units and 73460 amino acids for tree inference with Bayesian models.

Here, we focus on the second alignment with 136 taxa. We use shorter taxon names and I have converted the file from FASTA format to Phylip format which is required by Phylobayes (Lartillot et al. 2013). I provide the alignment `strassert-136taxa.phy` in the data subfolder of this tutorial:

```
data/step1-strassert-136taxa.phy
```

For the phylogenetic inference in the next step, we also need an unrooted topology. I have removed root of the tree in Figure 3 in Strassert et al. (2021). Similar to the alignment, I provide the unrooted topology in the data subfolder of this tutorial:

```
data/step1-strassert-136taxa.unrooted.tre
```

Please also see the README specifying the license and availability of the data.

4.2 Step 2: Phylogenetic inference

Here, we use a Bayesian method to estimate the posterior distribution. In particular, we will use Phylobayes (Lartillot et al. 2013), which you need to install (either the sequential and parallel version). In this step, we have to decide on an evolutionary model. The optimal evolutionary model is determined by the nature (for example, the time range between most recent common ancestor and the leaves of the tree) and the size of the data set (i.e., number of rows and columns in the alignment) which is an indicator for the computational requirements.

We identify evolutionary models using their exchangeabilities and possibly their across-site compositional heterogeneity (e.g., Koshi and Goldstein 1995) and rate heterogeneity (e.g., Yang 1993) models. One major advantage of McmcDate is the possibility to date trees with complex evolutionary models to infer the branch lengths measured in expected number of substitutions per site. In particular, we can use models accounting for across-site compositional heterogeneity which is important when dating trees over long evolutionary time scales (see, e.g., Szánthó et al. 2022). For example, the evolutionary model `GTR+CAT+G4` uses general time reversible exchangeabilities (GTR, Tavaré 1986), the Bayesian CAT model (Lartillot and Philippe 2004) to account for across-site compositional heterogeneity, and the gamma rate model (Yang 1993) with four components to account for across-site rate heterogeneity.

In general, I recommend the following evolutionary models sorted from preferred but more complex to fast but simpler: `GTR+CAT+G4`, `LG+CAT+G4` (LG: Le and Gascuel 2008), `LG+EDM64+G4` (EDM: Schrempf et al. 2020), `LG+C60+G4` (C60: Quang et al. 2008), and `LG+G4`. Phylobayes usually takes a long time to run, and I recommend using a dedicated environment such as a scientific cluster. Here, we will use the `LG+G4` model and the following script:

```
cat data/step2-phylobayes
```

```
#!/usr/bin/env sh
export afn=step1-strassert-136taxa.phy
export tfn=step1-strassert-136taxa.unrooted.tre
export md=lg

pb -d $afn -T $tfn -ncat 1 -$md -dgam 4 -x 10 10000 "${afn}_${tfn}_${md}+g4"
```

Phylobayes (or any other Bayesian phylogenetic software package) samples from the posterior distribution of trees, which is the input to McmcDate. I provide the corresponding tree list file for this analysis in the data directory:

```
head -n 3 data/step2-strassert-136taxa-lg+g4.treelist | sed -E "s/(.{80}).*$/\1 ... /"
echo " ... "
```

```
(((((((((((((((((Al_Alexand:0.0416858,Al_Neocera:0.0751388):0.125206,(Al_Crypthe ...
(((((((((((((((((Al_Alexand:0.0660346,Al_Neocera:0.0987786):0.0313159,(Al_Crypth ...
(((((((((((((((((Al_Alexand:0.0654659,Al_Neocera:0.0983142):0.0284921,(Al_Crypth ...
 ...
```

4.3 Step 3: Rooted topology and auxiliary data

Strassert et al. (2021) discuss one unrooted topology with two possible root positions. (1) The root separates amorphea from diaphoretickes and excavates (Figure 3 in Strassert et al. 2021), and (2) the root separates amorphea and excavates from diaphoretickes. Here, we choose option (1) because this tree also appears in the main text. In general, McmcDate is fast enough to allow analysis and comparison of more root positions. Further, we use auxiliary data from 33 fossil calibrations compiled by Strassert et al. (2021). Finally, the discussed order of endosymbiosis events enforce a set of node order constraints. Here, we use the hypothesis of Stiller et al. (2014), and direct transfers as discussed in Section 2.2.

I provide the rooted tree `step3-strassert-136taxa.rooted.tre`, the node age calibration file `step3-strassert-136taxa-calibrations.rooted.tre` in Newick tree format (Appendix B.1), and the node order constraint file `step3-strassert-136taxa-constraints-direct-stiller.csv` in CSV format (Appendix B.2) in the data directory.

4.4 Step 4: Dating with McmcDate

McmcDate is a Haskell program (Appendix C) and you need to install the Haskell toolchain. Luckily, installation is easy with GHCup. Follow the instructions on the Haskell website. I recommend using `cabal-install`, see also Appendix C.

Here, we will use a wrapper script around McmcDate. The wrapper script has a simple command-line user interface and also takes care of compiling the executable provided the Haskell toolchain has been installed properly. The wrapper script uses a configuration file which is a BASH script assigning some variables:

```
cat data/step4-strassert.conf
```

```
# Configuration of McmcDate nalysis for the Strassert dataset.

################################################################################
# Mandatory definitions.

# Analysis name (base file name of results).
analysis_name="strassert"
# File name of rooted tree in Newick format.
rooted_tree="step3-strassert-136taxa.rooted.tre"
# File name of trees estimated by a Bayesian method such as Phylobayes. Contains
# a list of Newick trees.
trees="step2-strassert-136taxa-lg+g4.treelist"

################################################################################
# Optional definitions.

# Calibrations. If the file ends with "csv", assume the calibrations are
# provided in CSV format. Otherwise, assume the calibrations are provided on a
# Newick tree (McmcTree specification; only L, U, and B are supported).
calibrations="step3-strassert-136taxa-calibrations.rooted.tre"

# Constraints in CSV format.
constraints="step3-strassert-136taxa-constraints-direct-stiller.csv"

# # Braces in JSON format.
#
# braces="braces.json"

# # Finally, a suffix can be provided to distinguish between results
# # directories.
#
# suffix="test"
```

Like so, we can execute McmcDate:

```
cd data
./step4-mcmcdate -f step4-strassert.conf -c -k ul s p
./step4-mcmcdate -f step4-strassert.conf -c -k ul s r
```

In brief: `-f` specifies the configuration file; `-c` and `-k` activate node age calibrations and node order constraints, respectively; `ul` uses an uncorrelated log normal relaxed molecular clock model; `s` uses a sparse covariance matrix when approximating the phylogenetic likelihood (Appendix A); the `p` command stands for *prepare*; and the `r` command stands for *run*. I have separated the preparation of input data from the actual MCMC inference so that we do not have to repeat the preparation step for subsequent MCMC inferences. For a full list of the options and arguments of the wrapper script, see Appendix C.1.

McmcDate is verbose in that it provides extensive logs to standard output as well as detailed output files in a directory named `results-calibrations-constraints-ulognormal-sparse`. The results directory varies according to the options used for the MCMC inference. A separate document describes the results of analyses with McmcDate.

This specific analysis takes around two hours on my personal laptop. Hence, one can date the tree using different auxiliary data files, rooted trees, or other internal settings (Appendix C.3). If you want to sample more points from the posterior function, you can continue the analysis:

```
./step4-mcmcdate -f step4-strassert.conf -c -k ul s c
```

If you repeat an analyses with similar parameters (but the same set of MCMC proposals), you can use the option `--init-from-save` (Appendix C.2).

4.5 Optional: Marginal likelihood calculation

After setting up an analysis like in Section 4.4, calculation of the marginal likelihood is as easy as:

```
cd data
./step4-mcmcdate -f step4-strassert.conf -c -k ul s m
```

The above command uses stepping stone sampling along 128 points to calculate the marginal likelihood. McmcDate is fast, but even so, the above command takes around 4 days on the computers I have used.

A Approximation of the phylogenetic likelihood

For a specific unrooted topology, McmcDate uses (an estimation of) the posterior distribution of branch lengths measured in expected number substitutions per site. The estimation of this posterior distribution happens in an external program as described in the main text. Now, McmcDate provides three ways to approximate the phylogenetic likelihood for a given set of branch lengths measured in expected number of substitutions per site:

Univariate approach For each branch, use a normal distribution with mean and variance extracted from the posterior function estimate. Approximate the phylogenetic likelihood with the product of the probability densities of all normal distributions evaluated at the given branch lengths, respectively. Let n be the number of branches. The algorithmic complexity of the univariate approach is linear: $O(n)$.

Full covariance matrix approach Estimate a covariance matrix from the posterior function estimate. Use a multivariate normal distribution with this covariance matrix to approximate the phylogenetic likelihood. For this to work, the covariance matrix has to be positive definite. The maximum likelihood estimator of the covariance matrix is positive definite, but we have to be careful when using sparse approaches (see below). The algorithmic complexity of this approach is quadratic: $O(n^2)$. That is, for large trees with thousands of branches this approach is too slow, and we have to approximate the approximation :).

Sparse covariance matrix approach Estimate a sparse inverted covariance matrix (precision matrix) using a graphical lasso (Friedman et al. 2007). The graphical lasso penalizes non-zero elements of the precision matrix. A parameter defines the penalization strength. After careful optimization using different statistics (e.g., the number of non-zero covariances), I chose a penalization parameter of 0.1. The penalization parameter can be specified freely when directly running McmcDate (Appendix C.2). I also tried other sparse matrix approaches but all suffered from estimating matrices that are not positive definite. I had implemented a subsequent step finding the positive definite matrix closest to the estimated sparse covariance matrix. However, I experienced worse results and have subsequently removed this feature in favor of the graphical lasso.

B Auxiliary data specifications

The specifications match McmcDate version 1.0.0.0 and may change between different versions of McmcDate.

B.1 Node age calibrations

Node age calibrations can be provided in two ways:

- with comma separated value (CSV) files, or
- with Newick tree files (MCMCTree specification; see the documentation of MCMCTree; only L, U, and B are supported).

If the filename ends with `csv`, assume the calibrations are provided in CSV format. Otherwise, assume the calibrations are provided on a Newick tree. The CSV file has a header (see below), and one or more rows of the following format:

```
Name,LeafA,LeafB,YoungAge,YoungProbabilityMass,OldAge,OldProbabilityMass
```

In this case, the calibrated node is uniquely defined as the most recent common ancestor of `LeafA` and `LeafB`. The age of the node is calibrated between the lower (young) and upper (old) boundary. The probability mass describes the softness (or hardness) of a boundary. In other words, the probability mass describes the steepness of the decline of the prior function outside the calibration interval. In general, the larger the probability mass the softer the boundary. We specify probability masses with respect to a normalized time interval of size 1.0. That is, probability masses have to be strictly positive and strictly less than 1.0, which is the total probability mass in the unit interval.

A case study: Assume the root has an age of 4.5 Gya. Then, the complete time interval from present (0 Gya) to the position of the root has a probability mass of 1.0. In this case, a probability mass value of 1×10^{-4} roughly corresponds to a time interval of $4.5\ \text{Gy} \cdot 1 \times 10^{-4} = 0.45\ \text{My}$. However, we attach halves of normal distributions to the uniform node age calibration intervals, and so the prior function at this specific boundary will decline to small values (a bit) faster than within 0.45 My. Of note, if the root is younger, for example, at 2.5 Gya, then a value of 1×10^{-4} is stricter in terms of absolute time units, and roughly corresponds to an interval of 0.25 My.

I usually use values between 1×10^{-4} (hard) and 3×10^{-2} (soft). If unsure, use probability masses of 2.5×10^{-2}, which corresponds to 2.5 percent probability at each boundary or constraint. A probability mass close to 1.0 will correspond to a prior function too soft to have any effect. Note that this way of specifying boundary softness using relative values independent of the actual node ages differs from MCMCTree which uses absolute values (Yang and Rannala 2005). When using a Newick tree to specify node age calibrations, and when no probability masses are provided, a default value of 1×10^{-2} is used. This measure is in place to support the same input files as MCMCTree.

To specify one-sided node age calibrations, omit the other boundary and the corresponding probability mass. For example, the following file defines a node age calibration with a lower boundary at 1×10^6 time units (years in this case) with probability mass 2.5×10^{-2}:

```
Name,LeafA,LeafB,YoungAge,YoungProbabilityMass,OldAge,OldProbabilityMass
Primates,Human,Chimpanzees,1e6,2.5e-2,,
```

B.2 Node order constraints

Node order constraints are provided using a comma separated value (CSV) file with a header (see below) and one ore more rows of the following format:

```
Name,YoungerLeafA,YoungerLeafB,OlderLeafA,OlderLeafB,ProbabilityMass
```

The younger and older nodes are uniquely defined as the most recent common ancestors of `YoungerLeafA` and `YoungerLeafB`, as well as `OlderLeafA` and `OlderLeafB`, respectively. As described in the previous section about node age calibrations, the probability mass describes the softness (or hardness) of the constraint. For example, the following file defines a constraint where the ancestor of leaves `A` and `B` is younger than the ancestor of leaves `C` and `D`:

```
Name,YoungerLeafA,YoungerLeafB,OlderLeafA,OlderLeafB,ProbabilityMass
ExampleConstraint,A,B,C,D,0.025
```

McmcDate reports and removes redundant constraints such as constraints affecting nodes that are vertically related.

B.3 Node braces

Node braces are provided using files in JavaScript object notation (JSON) format. Similar to node age calibrations and node order constraints, the braced nodes are specified using pairs of leaves. However, the softness (or hardness) of braces is defined in a different way. The reason is that more than two nodes can be braced, and so, there is no canonical way to describe the softness using probability mass. Rather, for a specific node brace, the differences between the node ages and the average age of all nodes in the particular node brace are normally distributed with the provided standard deviation.

The following example defines two node braces constraining two and three nodes, respectively:

```
[
  {
    "braceDataName": "Brace1",
    "braceDataNodes": [
      [
        "NodeXLeafA",
        "NodeXLeafB"
      ],
      [
        "NodeYLeafA",
        "NodeYLeafB"
      ]
    ],
    "braceDataStandardDeviation": 0.0001
  },
  {
    "braceDataName": "Brace2",
    "braceDataNodes": [
      [
        "NodeALeafA",
        "NodeALeafB"
      ],
      [
        "NodeBLeafA",
        "NodeBLeafB"
      ],
      [
        "NodeCLeafA",
        "NodeCLeafB"
      ]
    ],
    "braceDataStandardDeviation": 0.0001
  }
]
```

In this case, the braced nodes of the first node brace are uniquely defined as the most recent common ancestors of `NodeXLeafA` and `NodeXLeafB`, as well as `NodeYLeafA` and `NodeYLeafB`. The steepness of the brace prior function is defined using the standard deviation. This file defines hard node braces.

C Installation and internals

I have written McmcDate in Haskell. The Haskell programming language is versatile, interesting, and leads to more maintainable code with fewer bugs when compared to other programming languages. Nevertheless, the tooling support is sometimes sub-optimal. Before running McmcDate you need to (1) install the Haskell toolchain and (2) compile the Haskell code.

1. Installing the Haskell toolchain: I recommend using GHCup, a cross platform script handling the installation process. Check the website for instructions. Note, there are two build tools commonly used with Haskell: `cabal-install`, and Haskell `Stack` with binaries `cabal`, and `stack`, respectively. I recommend using `cabal-install` (see below).

2. Compiling the source code: Clone the source code of McmcDate:

   ```
   git clone git@github.com:dschrempf/mcmc-date.git
   ```

   Perform your analysis within the folder of the McmcDate repository. For example, run test analyses or the tutorial analysis from within the `mcmc-date/tests` or `mcmc-date/tutorial/data` directory, respectively. In this case, the wrapper script, which is used in this tutorial, compiles and runs the source code for you in a reproducible way, and so there is no need for manual action. Sometimes, however, manual action may be required. In this case, you need a rough understanding of the tools involved. For example, the wrapper script uses `cabal-install` by default. If you want to use `Stack`, use the option `-s` like so: `run -s` … . See also the output of run -h. In rare occasions you may want to clean your local build cache. You can do this by running cabal clean or stack clean; or more strictly, by deleting the dist-newstyle or .stack-work directories in you working directories for cabal-install and Stack, respectively.

C.1 Wrapper script

The wrapper script tries to make a good compromise between usability and customizability. It exposes some, but not all functionality of McmcDate:

```
Usage: step4-mcmcdate [OPTIONS] RELAXED_MOLECULAR_CLOCK_MODEL LIKELIHOOD_SPECIFICATION
↪  COMMANDS

Auxiliary data options:
-b Activate braces
-c Activate calibrations
-k Activate constraints

Algorithm related options:
-i NAME  Initialize state and cycle from previous analysis with NAME
-H       Activate Hamiltonian proposal (slow, but great convergence)
-m       Use Mc3 algorithm insteahd of Mhg

Other options:
-f FILE    Use a different analysis configuration file (relative path)
-n SUFFIX  Use an analysis suffix
-p         Activate profiling
-s         Use Haskell stack instead of cabal-install

Relaxed molecular clock model:
ug  Uncorrelated gamma model
ul  Uncorrelated log normal model
al  Autocorrelated log normal model

Likelihood specification:
f  Full covariance matix
s  Sparse covariance matrix
u  Univariate approach
n  No likelihood; use prior and auxiliary data only

Available commands:
p  Prepare analysis
r  Run dating analysis
c  Continue dating analysis
m  Compute marginal likelihood

A configuration file "analysis.conf" is required.
For reference, see the sample configuration file.
```

If you need to adjust specific parameters or settings, you can (a) call the McmcDate executable directly, and, if this is not enough, (b) directly change parameters or functions in the code. In the following, I briefly explain both options.

C.2 Direct invocation of McmcDate

Use the build tool of your choice (see above) to directly run McmcDate. For example, with cabal-install:

```
cabal run mcmc-date-run -- -h
```

```
Up to date
mcmc-date; version 1.0.0.0

Usage: mcmc-date-run COMMAND

  Date a phylogenetic tree using calibrations and constraints

Available options:
  -h,--help                Show this help text

Available commands:
  prepare                  Prepare data
  run                      Run MCMC sampler
  continue                 Continue MCMC sampler
  marginal-likelihood      Calculate marginal likelihood
```

The help shows that McmcDate exposes four sub-commands. For example, to get help about how to run a new analysis:

```
cabal run mcmc-date-run -- run -h
```

Hence, an example command line is:

```
cabal run -- mcmc-date-run run \
  --analysis-name example \
  --calibrations "csv calibrations.csv" \
  --constraints "constraints.csv" \
  --relaxed-molecular-clock "UncorrelatedLogNormal" \
  --likelihood-spec "SparseMultivariateNormal 0.1"
```

Get help about how to continue an analysis with:

```
cabal run mcmc-date-run -- continue -h
```

Consequently, continue the above example analysis with:

```
cabal run -- mcmc-date-run continue \
  --analysis-name example \
  --calibrations "csv calibrations.csv" \
  --constraints "constraints.csv" \
  --relaxed-molecular-clock "UncorrelatedLogNormal" \
  --likelihood-spec "SparseMultivariateNormal 0.1"
```

The commands are verbose. In my experience, detailed specification of the parameters and settings on the command line involves more investment in the beginning, but reduces the number of bogus analyses in the end.

C.3 Understanding the Haskell code

At the core of McmcDate are two libraries I have authored: mcmc, a general purpose Markov chain Monte Carlo (MCMC) sampler with advanced algorithms; and elynx-tree, a library for handling trees.

Additionally, I have separated McmcDate into two parts. Part (a) is the executable `mcmc-date-run` with modules specifying the state space, the prior and likelihood functions, the proposals, and MCMC-specific settings such as the number of burn-in and normal iterations. These modules are in the subfolder `app` of the McmcDate repository. Part (b) is a library containing prior functions and proposals specific to phylogenetic trees. I will not provide details for the library part here, but feel free to contact me for specific questions if you want. In general, you can access detailed help by rendering and opening the documentation directly contained in the source files:

```
cabal haddock mcmc-date
```

```
Build profile: -w ghc-9.2.4 -O1
In order, the following will be built (use -v for more details):
 - mcmc-date-1.0.0.0 (lib) (ephemeral targets)
Preprocessing library for mcmc-date-1.0.0.0..
Running Haddock on library for mcmc-date-1.0.0.0..
 100% (  4 /  4) in 'Mcmc.Tree.Import'
 100% ( 16 / 16) in 'Mcmc.Tree.Types'
 100% (  2 /  2) in 'Mcmc.Tree.Prior.Branch'
 100% ( 10 / 10) in 'Mcmc.Tree.Prior.Branch.RelaxedClock'
 100% (  6 /  6) in 'Mcmc.Tree.Prior.BirthDeath'
 100% (  3 /  3) in 'Mcmc.Tree.Monitor'
 100% ( 11 / 11) in 'Mcmc.Tree.Lens'
 100% ( 18 / 18) in 'Mcmc.Tree.Prior.Node.Constraint'
 100% ( 20 / 20) in 'Mcmc.Tree.Prior.Node.Calibration'
 100% (  2 /  2) in 'Mcmc.Tree.Prior.Node.CalibrationFromTree'
 100% ( 12 / 12) in 'Mcmc.Tree.Prior.Node.Brace'
 100% (  2 /  2) in 'Mcmc.Tree.Prior.Node.Combined'
 100% ( 12 / 12) in 'Mcmc.Tree.Proposal.Unconstrained'
 100% (  7 /  7) in 'Mcmc.Tree.Proposal.Ultrametric'
 100% (  7 /  7) in 'Mcmc.Tree.Proposal.Contrary'
 100% (  3 /  3) in 'Mcmc.Tree.Proposal.Brace'
 100% ( 22 / 22) in 'Mcmc.Tree'
Documentation created:
/home/dominik/Shared/haskell/mcmc-date/dist-newstyle/build/x86_64-linux/ghc-9.2.4/…
```

In my case, the documentation in HTML format is then available at

`${SUBTITUTE_ABOVE_PATH}/mcmc-date-1.0.0.0/doc/html/mcmc-date/index.html`

which, I am sure, you can remember easily. Did I already say that Haskell tooling has room for improvements?
Now, back to the application part (a), which will most likely be more important for you. In particular, you may want to have a look at the modules

Definitions Contains proposals and monitors, as well as MCMC-specific settings. If you want to change the number of burn-in iterations, or the number of total iterations, have a look and change this file.

State Defines the state space. If you really want to understand what is going on, this should be your starting point. The documentation is detailed, and explains the separation of the time tree and rate tree objects, as well as the birth and death prior.

The other more important modules are:

Main Contains functions to prepare the data, as well as to run and continue the analysis. This module also contains helper functions to calculate the marginal likelihood.

Probability Defines the prior and likelihood functions. This module is important, if you want to tweak the prior.

The less important modules are:

Hamiltonian Hamiltonian proposal.

Monitor Prior specific monitoring functions.

Options Handle command line options.

Tools Miscellaneous tools.

References

Friedman, J., T. Hastie, and R. Tibshirani (2007). “Sparse inverse covariance estimation with the graphical lasso.” In: *Biostatistics* 9.3, pp. 432–441. DOI: 10.1093/biostatistics/kxm045.

Koshi, J. M. and R. A. Goldstein (1995). “Context-dependent optimal substitution matrices.” In: *Protein Engineering Design and Selection* 8.7, pp. 641–645. DOI: 10.1093/protein/8.7.641.

Lartillot, N. and H. Philippe (2004). “A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process.” In: *Molecular Biology and Evolution* 21.6, pp. 1095–1109. DOI: 10.1093/molbev/msh112.

Lartillot, N., N. Rodrigue, D. Stubbs, and J. Richer (2013). “PhyloBayes MPI: Phylogenetic Reconstruction with Infinite Mixtures of Profiles in a Parallel Environment.” In: *Systematic Biology* 62.4, pp. 611–615. DOI: 10.1093/sysbio/syt022.

Le, S. Q. and O. Gascuel (2008). “An improved general amino acid replacement matrix.” In: *Molecular Biology and Evolution* 25.7, pp. 1307–1320. DOI: 10.1093/molbev/msn067.

Minh, B. Q., H. A. Schmidt, O. Chernomor, D. Schrempf, M. D. Woodhams, A. von Haeseler, and R. Lanfear (2020). “IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era.” In: *Molecular Biology and Evolution* 37.5, pp. 1530–1534. DOI: 10.1093/molbev/msaa015.

Quang, L. S., O. Gascuel, and N. Lartillot (2008). “Empirical profile mixture models for phylogenetic reconstruction.” In: *Bioinformatics* 24.20, pp. 2317–2323. DOI: 10.1093/bioinformatics/btn445.

Reis, M. dos and Z. Yang (2011). “Approximate Likelihood Calculation on a Phylogeny for Bayesian Estimation of Divergence Times.” In: *Molecular Biology and Evolution* 28.7, pp. 2161–2172. DOI: 10.1093/molbev/msr045.

Schrempf, D., N. Lartillot, and G. Szöllősi (2020). “Scalable empirical mixture models that account for across-site compositional heterogeneity.” In: *Molecular Biology and Evolution*. DOI: 10.1093/molbev/msaa145.

Stiller, J. W., J. Schreiber, J. Yue, H. Guo, Q. Ding, and J. Huang (2014). “The evolution of photosynthesis in chromist algae through serial endosymbioses.” In: *Nature Communications* 5.1. DOI: 10.1038/ncomms6764.

Strassert, J. F. H., I. Irisarri, T. A. Williams, and F. Burki (2021). "A molecular timescale for eukaryote evolution with implications for the origin of red algal-derived plastids." In: *Nature Communications* 12.1. DOI: 10.1038/s41467-021-22044-z.

Szánthó, L. L., N. Lartillot, G. J. Szöllősi, and D. Schrempf (2022). "Compositionally constrained sites drive long branch attraction." In: *bioRxiv*. DOI: 10.1101/2022.03.03.482715.

Szöllõsi, G. J., S. Höhna, T. A. Williams, D. Schrempf, V. Daubin, and B. Boussau (2022). "Relative Time Constraints Improve Molecular Dating." In: *Systematic Biology* 71.4, pp. 797–809. DOI: 10.1093/sysbio/syab084.

Szöllősi, G. J., E. Tannier, N. Lartillot, and V. Daubin (2013). "Lateral Gene Transfer from the Dead." In: *Systematic Biology* 62.3, pp. 386–397. DOI: 10.1093/sysbio/syt003.

Tavaré, S. (1986). "Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences." In: *Lectures on Mathematics in the Life Sciences* 17, pp. 57–86.

Yang, Z. (1993). "Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites." In: *Molecular Biology and Evolution*. DOI: 10.1093/oxfordjournals.molbev.a040082.

Yang, Z. and B. Rannala (2005). "Bayesian Estimation of Species Divergence Times Under a Molecular Clock Using Multiple Fossil Calibrations with Soft Bounds." In: *Molecular Biology and Evolution* 23.1, pp. 212–226. DOI: 10.1093/molbev/msj024.

Yang, Z. (2006). *Computational molecular evolution.* Oxford Series in Ecology and Evolution. Oxford University Press.