sMap

Evolution of independent, dependent and conditioned discrete characters in a Bayesian framework

Giorgio Bianchini, Patricia Sánchez-Baracaldo

giorgio.bianchini@bristol.ac.uk

Contents

1 Introduction

sMap is a program to perform stochastic mapping (Nielsen, 2002; Huelsenbeck, Nielsen and Bollback, 2003) analyses on discrete characters. This kind of analysis involves estimating substitution parameters, reconstructing ancestral states and simulating histories, in order to study the evolution of multiple types of discrete characters (e.g. morphological features, presence of genes, habitats...), without necessarily relying on a single phylogenetic tree.

Unlike other stochastic mapping software, such as SIMMAP (Bollback, 2006) and the `make.simmap` function in the R package `phytools` (Revell, 2012), sMap implements a wide range of prior distributions for parameters, multiple evolutionary models, and three different kinds of characters: *independent* characters, *dependent* characters, and *conditioned* characters.

Independent characters are analysed through a continuous-time Markov chain model – i.e. some parameters ("rates") are estimated and then used to simulate the evolution of these characters, under the assumption that their history does not have any effect on other independent characters. For example, the appearance of two completely unrelated morphological features may be modelled as two independent characters.

Sets of *dependent* characters are treated as a single unit (that evolves according to a continuous-time chain Markov model): this way, it is possible to consider effects that link the evolution of different characters. For example, the presence of two related genes belonging to the same biochemical pathway might be modelled as two dependent characters.

Finally, *conditioned* characters do not follow a continuous-time Markov chain model: their state is instead completely determined by the state of other characters, which are "conditioning" the conditioned characters. For example, the ability to produce a certain molecule may be modelled as a character that is conditioned on the presence of the genes involved in the pathway to produce said molecule (those genes would also, in turn, be dependent on each other).

By combining these different types of characters, it is possible to construct complicated models that describe multiple aspects of a certain feature of evolutionary interest.

1.1 A brief introduction to stochastic mapping

For a detailed description of the stochastic mapping, please see (Huelsenbeck, Nielsen and Bollback, 2003).

A stochastic mapping analysis is used to reconstruct the evolutionary history of a discrete character on a phylogenetic tree, not only at the nodes of the tree, but also along each branch. Given the appropriate priors and data, it allows, in a Bayesian framework, to assess the posterior probability that a character has of having been in each possible state, at any point on a branch of the tree.

This analysis essentially involves the following five steps:

1. Parameter estimation/sampling
2. Ancestral state reconstruction
3. Sampling of internal node states
4. Simulation
5. Summarisation of results

These steps are described below and analysed in more detail further down.

Before carrying out a stochastic mapping analysis, it is necessary to decide how many simulations to perform: a higher number of simulations will require a longer execution time, but will result in more accurate conclusions; conversely, a lower number of simulations will take a short time, but the estimates may be rougher. This number will be referred to as n in the following.

The stochastic mapping process is summarised in **Figure 1**.

1.1.1 Parameter sampling

The first step in the analysis is to determine the values of the parameters of the evolutionary model that is used to analyse the data. The number of parameters to estimate varies according to the chosen model, and there are multiple ways in which they can be estimated (such as, Maximum Likelihood or a Bayesian approach). In any case, the result of this step will be n sample sets of parameters (which may all be identical, if wanted).

Each sample set of parameters is estimated according to a phylogenetic tree. It is however not necessary that the same tree be used to estimate all n samples: multiple trees can be used and lead to different estimates (e.g. the posterior distributions of the parameters conditioned on different topologies will be different).

Under a Maximum Likelihood approach, the downside to this is that the estimation needs to be performed separately for each tree, thus the computational time increases linearly with the number of trees (even when they only differ for the branch lengths). However, this problem does not occur when using a Bayesian approach to sample model parameters.

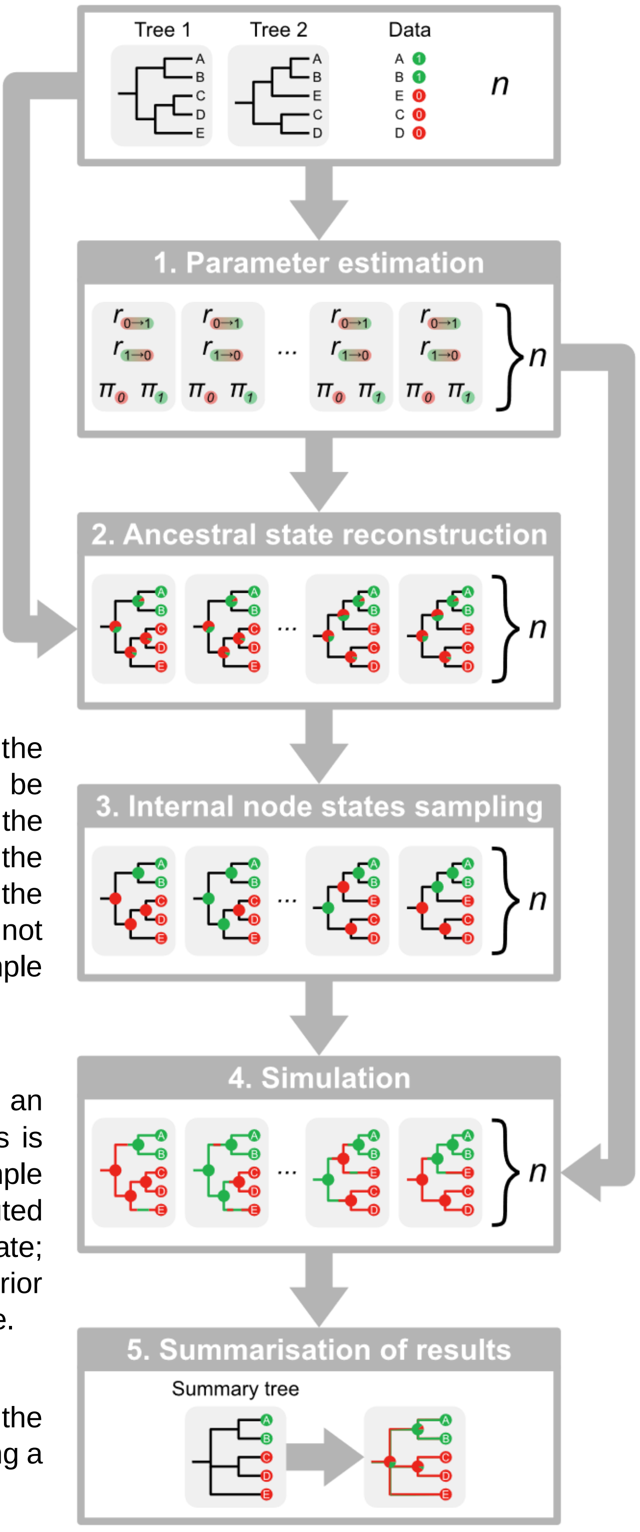


Figure 1. Summary of the stochastic mapping process.

1.1.2 Ancestral state reconstruction

For each of the n samples of parameters, an ancestral state reconstruction of the node states is performed (on the same tree that was used to sample the parameters). Priors and likelihoods are computed for each node in the tree and for each possible state; they are then combined to yield the posterior probabilities that each node was in a certain state.

1.1.3 Sampling of internal node states

Using the posterior distributions computed in the previous step, n histories are sampled by choosing a state for each node.

1.1.4 Simulation

n continuous-time Markov chain simulations are run, one for each of the histories sampled in the previous step and the parameters sampled in step 1. As the evolution along each branch is simulated, simulations that are not consistent with the sampled history are discarded and re-run, in order to condition the evolution of the character on the sampled history.

1.1.5 Summarisation of results

If multiple different trees have been used to run the simulations, a single summary tree (which could be, e.g., a consensus tree) is needed for this step: this tree will then be used to report the summarised results.

For each point in time, the posterior distribution of the state of every branch in the tree at that time will be computed by averaging the simulated histories which include that branch at that point in time.

1.2 Evolutionary models

In order to simulate the evolution of a discrete character with a continuous-time Markov chain, an evolutionary model is needed, that specifies which states can change (*transition*) into which other states, and at what *rate* each transition happens. The model is thus specified by a transition rate matrix, which is a quasipositive square matrix with the diagonal elements being the negated sum of each row. These are complicated words to describe something that looks like this:

$$Q = \begin{bmatrix} -\sum_{i=2}^{n} q_{1i} & q_{11} & \cdots & q_{1n} \\ q_{21} & -\sum_{\substack{i=1 \\ i\neq 2}}^{n} q_{2i} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & -\sum_{i=1}^{n-1} q_{ni} \end{bmatrix} \qquad q_{ij} \geq 0 \; \forall \; i \neq j$$

Or, in a more concrete example:

$$Q_{JC} = \begin{bmatrix} -\frac{3}{4} & {}^{1}/_{4} & {}^{1}/_{4} & {}^{1}/_{4} \\ {}^{1}/_{4} & -\frac{3}{4} & {}^{1}/_{4} & {}^{1}/_{4} \\ {}^{1}/_{4} & {}^{1}/_{4} & -\frac{3}{4} & {}^{1}/_{4} \\ {}^{1}/_{4} & {}^{1}/_{4} & {}^{1}/_{4} & -\frac{3}{4} \end{bmatrix}$$

Models specified this way are not exclusive to stochastic mapping algorithms, but are also used, e.g., when using alignments to build phylogenetic trees: case in point, Q_{JC} above specifies a Jukes-Cantor model.

The size s of the matrix (i.e. the number of rows and columns) is equal to the number of possible states of the character being analysed (e.g., 4 for a DNA nucleotide, 2 for a presence/absence character, etc), and the diagonal entries in the matrix above have been highlighted in grey, because they are completely determined by the other entries (and are often not considered/reported when describing a model). This means that this kind of model can have at most $s \cdot (s - 1)$ parameters (e.g., 12 for a DNA nucleotide[1], 2 for a presence/absence character etc).

[1] The reader may be aware that the "most general" DNA evolution model, the Generalised time-reversible (GTR) model (Tavaré, 1986), only has 9 parameters (6 transition rates and 3 "equilibrium frequencies"). This

Each element q_{ij} of the matrix (except for the diagonal entries) is related to the transition from state i to state j, and gives the rate for that transition. For example, if this number is close to zero, this means that a character that is found in state i will be unlikely to shift into state j.

Furthermore, to fully define the continuous-time Markov chain, it is necessary to specify the probability distribution of the states at the start of the process (the root of the tree), i.e. the probabilities π_i that the root node was in each state i[2], called the “root node priors”.

1.2.1 Evolutionary models implemented in sMap

sMap allows the user to fully specify the evolutionary model in this framework. It is possible to set each element of the matrix to a fixed value (including 0, to disallow certain transitions), to allow it to be estimated by the program using Maximum Likelihood, to specify a prior to estimate it with Bayesian inference, or to constrain it to be equal to another element of the matrix (**Figure 2**).

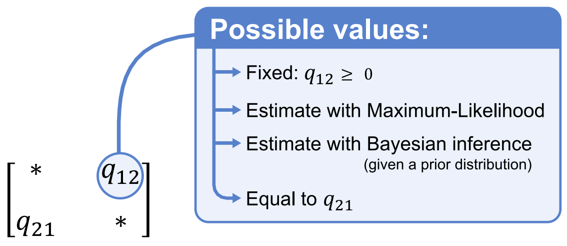

Possible values:
Fixed: $q_{12} \geq 0$
Estimate with Maximum-Likelihood
Estimate with Bayesian inference
(given a prior distribution)
Equal to q_{21}
$\begin{bmatrix} * & q_{12} \\ q_{21} & * \end{bmatrix}$

Figure 2. Possible values for a parameter of a two-state evolutionary model

1.2.2 Examples of evolutionary models

This way, it is possible to implement many different evolutionary models.

For example, take a character X with two possible states (let them be 1 and 2). These are all the possible models for this character (assuming that a and b are parameters that could be fixed to a user-supplied value greater than 0, or estimated using maximum likelihood, or estimated using Bayesian inference):

$$A = \begin{bmatrix} * & a \\ b & * \end{bmatrix} \qquad B = \begin{bmatrix} * & 0 \\ b & * \end{bmatrix} \qquad C = \begin{bmatrix} * & a \\ a & * \end{bmatrix}$$

$$D = \begin{bmatrix} * & a \\ 0 & * \end{bmatrix} \qquad E = \begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix}$$

In model A, the character can change from state 1 to state 2 with rates $r(1 \to 2) = a$ and $r(2 \to 1) = b$, respectively.

In model B, instead, the character can change from state 2 to state 1 with rate $r(2 \to 1) = b$, but it cannot change from state 1 to state 2, since $r(1 \to 2) = 0$. This means that, according to this model, after a sufficiently long evolutionary time, the character will be in state 1 in every taxon (regardless of the root state).

In model C, the character can change back and forth between the two states, with equal rates: $r(1 \to 2) = r(2 \to 1) = a$. After a sufficiently long time, each taxon will have a 50% probability of being in either state 1 or 2.

Model D is similar to model B, in which it allows only one type of transition (from state 1 to state 2); accordingly, after a sufficiently long time, the character will be in state 2 in every taxon.

---

is in fact because the GTR model is a *time-reversible* model, unlike the models used in stochastic mapping. Intuitively, a time-reversible model is useful if one wants to be agnostic towards the root of the phylogeny (as the likelihood will not vary with root placement), while non-time-reversible models can be used when a reliable rooting is provided.

[2] Please note that, even though the symbol π is used, these probabilities are not related to the “equilibrium frequencies” of time-reversible models.

Finally, model E does not allow any kind of transitions: therefore, each taxon will have the same state as the root node.

While it is possible to enumerate all possible models for a two-state character, the number of models grows exponentially very quickly with the number of possible character states: already for a three-state character there are tens of thousands of possible models. We will thus now concentrate and describe a few model "categories", but note that it is entirely possible to define other models and use them in sMap.

1.2.2.1 All-rates-different (ARD) models

All-rates-different (ARD) models, as the name implies, allow any kind of transition between states, with a different rate for each transition. These are the most parameter-rich models, as they have $s \cdot (s-1)$ different parameters. The matrix on the right shows a transition rate matrix for an ARD model for a three-state character (an ARD model for a two-state character is model A above).

$$\begin{bmatrix} * & a & b \\ c & * & d \\ e & f & * \end{bmatrix}$$

1.2.2.2 Equal-rates (ER) models

In equal-rates (ER) models, any transition is still allowed, but they all happen with the same rate. These are the least parameter-rich models, as they only have a single parameter. The matrix on the right shows a transition rate matrix for an ER model for a three-state character (an ER model for a two-state character is model C above).

$$\begin{bmatrix} * & a & a \\ a & * & a \\ a & a & * \end{bmatrix}$$

1.2.2.3 Symmetrical (SYM) models

In a symmetrical (SYM) model, any transition is still allowed, but each transition happens at the same rate of the reverse transition, which can be different from other transitions (e.g. $r(1 \to 3) = r(3 \to 1)$ and $r(2 \to 3) = r(3 \to 2)$, but $r(1 \to 3)$ and $r(2 \to 3)$ can be different). The resulting matrix is symmetrical with respect to the diagonal. These models are middle-of-the-ground in terms of parameter-richness, as they have $\frac{s \cdot (s-1)}{2}$ parameters. The matrix on the right shows a transition rate matrix for a SYM model for a three-state character (for a two-state character, a SYM model and an ER model are indistinguishable).

$$\begin{bmatrix} * & a & b \\ a & * & c \\ b & c & * \end{bmatrix}$$

1.2.2.4 Ordered (ORD) models

When there are more than two possible character states, an ordered model (ORD) only allow transitions between certain states, forcing the evolution of the character to happen in a certain "order". For example, the matrix on the right disallows the direct transitions between states 1 and 3, thus for the character to go from state 1 to state 3 or vice versa, it must pass through state 2 ($1 \leftrightarrow 2 \leftrightarrow 3$). It is of course also possible to have an ORD ER model (i.e. $a = b = c = d$) or an ORD SYM model (i.e. $a = b$ and $c = d$).

$$\begin{bmatrix} * & a & 0 \\ b & * & c \\ 0 & d & * \end{bmatrix}$$

1.2.3 Model selection

It is very important to choose an appropriate evolutionary model for a stochastic mapping analysis, as the results may change dramatically with different models. There are multiple ways to perform model selection that can be used with sMap.

When a maximum-likelihood estimate analysis is done, the value of the maximum likelihood can be used to perform tests using the **A**kaike **I**nformation **C**riterion (AIC, Akaike, 1998) and the **B**ayesian **I**nformation **C**riterion (BIC, Schwarz, 1978). These criteria take into account both the value of the maximum likelihood and the number of parameters for the different models and provide a way to choose the model that best fits the data without adding unnecessary parameters.

In a Bayesian context, it is also possible to choose between different models by comparing the models' marginal likelihoods and computing Bayes factors and posterior probabilities for the models.

1.2.3.1 *Akaike information criterion*

Given a model with k parameters and the maximum value for the likelihood for that model $\hat{L}$, the Akaike Information Criterion (AIC) is defined as:

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

The value of the AIC increases with k and decreases with $\hat{L}$, which means that between two models with the same maximum likelihood, the one with more parameters will have a higher AIC and, similarly, between two models with the same number of parameters, the one with a lower maximum likelihood will have a higher AIC.

When the sample size is small, a corrected version of the AIC is used (AICc – AIC corrected). The way the AICc is computed depends on the model; in sMap it is computed as:

$$\mathrm{AICc} = \mathrm{AIC} + \frac{2k^2 + 2k}{n - k - 1}$$

Where n (the sample size) is equal to $\text{number of taxa} \cdot \text{number of characters}$.

When comparing models with the AIC(c), the best model will be the one with the lowest value for the AIC(c). For each model i, let $\Delta_i = \mathrm{AIC(c)}_i - \mathrm{AIC(c)}_{min}$, where $\mathrm{AIC(c)}_{min}$ is the AIC(c) for the best model. A rule of thumb for comparing AIC(c) values is the following (Burnham and Anderson, 2004):

- $\Delta_i \leq 2$: substantial support for the model
- $4 \leq \Delta \leq 7$: considerably less support
- $\Delta_i > 10$: essentially no support

When there are multiple models with similar AIC(c) values, it is possible to compute Akaike weights:

$$w_i = \frac{\exp\left(-\frac{\Delta_i}{2}\right)}{\sum \exp\left(-\frac{\Delta_i}{2}\right)}$$

Note that it is the same to compute these weights using the Δ_i or using the raw $\mathrm{AIC(c)}_i$, but using the Δ_i helps to avoid numerical underflow problems. These weights are approximations of model posterior probabilities and can be used to perform model averaging of the results: an analysis is done for each model, and then the results are averaged using the Akaike weights. This can be done in sMap using the Blend-sMap utility.

When a maximum-likelihood analysis is performed, sMap reports values for both the AIC and AICc. Akaike weights should be computed by the user, though.

1.2.3.2 *Bayesian Information Criterion*

Like AIC(c), the Bayesian Information Criterion considers the number of parameters k in the model and the maximum value $\hat{L}$ of the likelihood for the model, but the penalising term also takes into account the sample size n of the data (computed in sMap as $\text{number of taxa} \cdot \text{number of characters}$):

$$BIC = k \cdot \ln(n) - 2\ln(\hat{L})$$

Just like with AIC(c), when comparing BIC values, the best model will be the one with the lowest BIC value. When comparing the BIC scores for two models, the following rules of thumb can be used (Raftery, 1995) to assess the difference Δ_{BIC} between the two models:

- $\Delta_{BIC} \leq 2$: weak evidence in favour of one of the models
- $2 < \Delta_{BIC} \leq 6$: positive evidence
- $6 < \Delta_{BIC} \leq 10$: strong evidence

- $\Delta_{BIC} > 10$: very strong evidence

BIC weights can be computed and used just like Akaike weights:

$$w_i = \frac{\exp\left(-\frac{\text{BIC}_i}{2}\right)}{\sum \exp\left(-\frac{\text{BIC}_i}{2}\right)}$$

When a maximum-likelihood analysis is performed, sMap also reports BIC values (but not BIC weights).

1.2.3.3 Bayes factors and model posterior probabilities

Given a model M, a set of parameters $\boldsymbol{\theta}$ for the model (which can be a vector including multiple parameters, e.g. transition rates and root prior probabilities), and some Data, the likelihood of the model and model parameters given the data is the probability of obtaining the data when the model is true and those parameters are used:

$$L(M, \boldsymbol{\theta} | \text{Data}) = \mathbb{P}(\text{Data} | M, \boldsymbol{\theta})$$

The *maximum* likelihood for model M is thus the value of $L(M, \boldsymbol{\theta} | \text{Data})$ with the model parameters $\boldsymbol{\theta}$ chosen as to obtain the largest possible value. The *marginal* likelihood is instead the overall probability $\mathbb{P}(\text{Data} | M)$ of obtaining the data when the model is true (taking into account all possible values of $\boldsymbol{\theta}$).

The marginal likelihood cannot be computed analytically in most cases; it can however be approximated using numerical (Monte Carlo) methods. A technique to compute marginal likelihoods that is implemented in sMap is the stepping-stone algorithm (Xie *et al.*, 2011).

Marginal likelihoods can be used to compute Bayes factors (BF) to compare two models:

$$BF = \frac{\mathbb{P}(\text{Data} | M_1)}{\mathbb{P}(\text{Data} | M_2)}$$

Where M_2 is the model with the smallest marginal likelihood (so that $BF \geq 1$). A rule of thumb for Bayes factors is the following (Raftery, 1995):

- $1 < BF \leq 3$: weak evidence in favour of M_1
- $3 < BF \leq 20$: positive evidence
- $20 < BF \leq 150$: strong evidence
- $BF > 150$: very strong evidence

However, marginal likelihoods can be used more effectively to compute posterior probabilities for the models. To do this, we need first of all a prior on the models $\mathbb{P}(M)$; this should depend on prior beliefs on the models, but it can in most cases be assumed to be uniform (i.e. have the same value for all models). In principle, it would be necessary to compute the marginal likelihood for all possible models, but this can be avoided by assuming that the prior probability is 0 for the models that have not been considered.

The posterior probability for a model M_i is:

$$\mathbb{P}(M_i | Data) = \frac{\mathbb{P}(Data | M_i) \cdot \mathbb{P}(M_i)}{\sum_j \mathbb{P}(M_j) \cdot \mathbb{P}(Data | M_j)}$$

Which, in the case of a uniform prior over the models, reduces to:

$$\mathbb{P}(M_i | Data) = \frac{\mathbb{P}(Data | M_i)}{\sum_j \mathbb{P}(Data | M_j)}$$

These posterior probabilities can be used as weights to average the results of the analyses (just as the AIC(c) and BIC).

When a stepping-stone analysis is performed in sMap, the value of the (natural) logarithm of the marginal likelihood is reported. The user can then compute the model posterior probabilities.

2 Input data

Regardless of the chosen evolutionary model(s), the following elements are necessary to perform a stochastic mapping analysis:

- A tree (or, a sample of trees)
- Character state data for the leaves (terminal nodes) of the tree

2.1 Trees

Trees can be provided to sMap in Newick format (Felsenstein, 1986). It is possible to provide either a tree file with a single Newick tree, or a file with multiple trees (one per line). These trees would ideally be posterior samples from a Bayesian phylogenetic analysis.

If multiple trees are used, it is also necessary to provide a single tree, which will be used to summarise the results (this could be, for example, a consensus tree with mean branch lengths).

All trees must be rooted and should also be clock-like (i.e. the length from the root to any of the tips should be the same). This latter requirement is not strictly necessary (the analysis will also work with non-clock-like trees), but it is encouraged to comply with it, as usually the only parameter that underlies the evolution of both the character under analysis and the sequences used to infer the phylogenetic tree is time (and tree topology).

It is not necessary, though, that the clock-like trees be time-calibrated: therefore, even if no data is available to calibrate a molecular clock, inferences conducted with an uncalibrated molecular clock can still be used.

2.2 Character state data

Character state data can be provided in a relaxed PHYLIP-like format. The first line of the input file should contain the number of taxa, one or more spaces (or tabulations), and the number of characters. Following this, one line per taxon should contain the taxon name, one or more spaces (or tabulations) and character data.

The character data that is supplied is interpreted as the *likelihood* of the state of the character for each taxon.

It can be provided in two ways:

- A single alphanumeric character (e.g. "A", "b", "1")
- A list of multiple state:likelihood couples, separated by commas and surrounded by curly brackets (e.g. "{A:0.5,B:0.5}")

If a single alphanumeric character is provided, then the likelihood that the character is in that state for that taxon is set to 1, and the likelihood that the character is in other states is set to 0. If instead the other syntax is used, the likelihood that the character is in each of the specified states is set to the specified value.

Data for multiple characters can be separated by spaces (or tabulations) or not.

This is an example of a valid input file, describing whether various groups of animals are e**N**dotherms or e**C**totherms (first character) and whether they have **F**eathers or n**O**t (second character). Please note that this is just meant to be an example, and is not necessarily 100% accurate. For “carnosaurs” (which are extinct dinosaurs), the uncertainty about these characters has been supplied.

```
7     2
Fish            C     O
Amphibians      C     O
Squamates       C     O
Crocodiles      C     O
Birds           N     F
Carnosaurs      {C:0.2,N:0.8}    {F:0.7,O:0.3}
Mammals         N     O
```

The program will interpret this file by associating the likelihoods for each taxon and for each character as follows:

	Character 1 (metabolism)		Character 2 (feathers)	
State	**C**	**N**	**O**	**F**
Fish	1	0	1	0
Amphibians	1	0	1	0
Squamates	1	0	1	0
Crocodiles	1	0	1	0
Birds	0	1	0	1
Carnosaurs	0.2	0.8	0.3	0.7
Mammals	0	1	1	0

Please note that the likelihoods do not need to sum to 1 (and, from a computational point of view, they are defined up to a multiplicative constant, so they do not need to be smaller than 1 either); what actually matters is the ratio between the various likelihoods for a certain character. For example, if the likelihoods for character 1 for Carnosaurs had been defined as 0.1/0.4 or as 0.4/1.6, the results of the analysis would have been exactly the same (the only difference would have been that the computed likelihoods would have been slightly different).

The exact form of these likelihoods depends on the character being analysed. For example, suppose we are analysing the **P**resence or **A**bsence of a gene G in a genome that has been sequenced with c completeness (e.g. $95\% = 0.95$) and x contamination (e.g. $3\% = 0.03$).

- First, suppose that we have done e.g. a BLAST (Camacho *et al.*, 2009) search, and we *did* find the gene. The likelihood that the character is in state **P** (i.e. “present”) is defined as: “the probability of finding the gene in the BLAST search, given that the gene is actually present in the genome”, which is equal to the probability that the gene is in the 95% of the genome that we have sequenced. Therefore:
  - $\mathbb{L}(G = P) = c = 0.95$

  Instead, the likelihood that the character is in state **A** (“absent”) is “the probability of finding the gene in the BLAST search, given that it is not actually present in the genome”, which is equal to the probability that the gene is in the 3% of genome contamination. Therefore:
  - $\mathbb{L}(G = A) = x = 0.03$

- Now, suppose instead that we *did not* find the gene in the BLAST search. Now, the likelihood that the character is in state **P** is “the probability of *NOT* finding the gene in a BLAST search, given that it is actually present in the genome”, which is equal to the probability that the gene falls in the 5% of the genome that we have not sequenced yet. Therefore
  - $\mathbb{L}(G = P) = 1 - c = 0.05$

  Instead, the likelihood that the character is in state **A** is “the probability of *NOT* finding the gene in a BLAST search, given that it is not actually present in the genome”, which is equal to the probability that the gene is not found in the 3% of the genome contamination. Therefore:

- $\mathbb{L}(G = A) = 1 - x = 0.97$

This is a simple example[3], but it shows how this way of specifying likelihoods can be powerful in incorporating uncertainty about the underlying data. Also note that when the genome is completely sequenced ($c = 1$) and there is no contamination ($x = 0$), the likelihoods revert to the familiar 0/1 that can be specified by a single letter in the input file.

2.3 Evolutionary models

Evolutionary models for the characters specify three different kinds of information: the relationships between characters ("dependencies"), the evolutionary rates ("rates") and the root node priors ("pis"). These kinds of information can be provided in different NEXUS files (Maddison, Swofford and Maddison, 1997) or can be combined in a single file. All NEXUS files are characterised by a header (`#NEXUS`) and one or more data blocks (which can be used to specify dependencies, rates or pis).

2.3.1 Dependencies

A dependency block specifies whether characters are independent, dependent or conditioned on each other. In the case of dependent characters, the rates and pis are also specified in the dependency block. For conditioned characters, the conditioned probabilities are also specified in this block.

An example of a NEXUS file containing a dependency block is the following:

```
#NEXUS

Begin Dependency;
    Independent: 0;

    Dependent: 1, 2;
        Pis:
            Default: Dirichlet(1)
            ;

        Rates:
            Default: Exponential(100)
            0,0 > 1,0: LogNormal(-2.3016, 1)
            0,1 > 1,1: LogNormal(1.9741, 1)
            1,0 > 0,0: LogNormal(-1.2036, 1)
            ;

    Conditioned: 3 | 0, 1, 2;
        Default: Dirichlet(1);
        Probs:
            0,0,0 > 0: 1
            1,1,1 > 1: 1
        ;

End;
```

[3] A couple of assumptions are made here to be able to work with the contamination in a simple way: any contaminant genome should be about the same size as the sequenced genome, and every contaminant genome should contain the gene G. These assumptions can be relaxed by conveniently changing how the likelihoods for state **A** are computed in the two cases, but this is out of the scope of this document.

This file pertains to a dataset that contains data for 4 characters, each of which can have two states (`0` and `1`). Let's examine the file line by line:

- Line 1: NEXUS file header.
- Lines 3 and 25: these lines mark the start ("`Begin Dependency;`") and end ("`End;`") of the dependency block.
- Line 4: this line specifies that the first character (i.e. character 0) evolves independently from the others.
- Line 6: this line specifies that characters 1 and 2 depend on each other. They are thus merged into a single "supercharacter" which has states `0,0`, `0,1`, `1,0` and `1,1`.
- Lines 7 and 9: these lines mark the start and end of the specification of the pis for the dependent characters.
- Line 8: this line specifies that the pis should be estimated using Bayesian sampling, with a flat Dirichlet prior.
- Lines 11 and 16: these lines mark the start and end of the specification of the rates for the dependent characters.
- Line 12: this line specifies that except for the rates that are addressed in the following lines, the rates should be estimated using Bayesian sampling with an exponential prior with $\lambda = 100$.
- Lines 13, 14 and 15: these lines specify that for the mentioned rates LogNormal priors should be used.
- Line 18: this line specifies that character 3 is conditioned on characters 0, 1 and 2.
- Line 19: this line specifies that except for the ones that are addressed in the following lines, the conditioned probabilities should be estimated using Bayesian sampling with a flat Dirichlet prior.
- Lines 20 and 23: these lines mark the start and end of the specification of the conditioned probabilities for the conditioned character.
- Lines 21 and 22: these lines specify the conditioned probabilities. Note that these are referred to as `conditioning_state > conditioned_state` (e.g. $\mathbb{P}(A|B)$ becomes `B > A`).

2.3.2 Rates

A rate block specifies the transition rate matrix for independent characters.

This is an example of a NEXUS file containing a rate block:

```
#NEXUS

Begin Rates;
    Character: 0;
    Rates:
        A > B: LogNormal(0.49487866517724, 1)
        B > A: LogNormal(0.0539564537713429, 1)
    ;
End;

Begin Rates;
    Character: 1;
    Rates:
        0 > 1: Exponential(100)
        1 > 0: LogNormal(0.743602233221588, 1)
    ;
End;
```

This file pertains to a dataset that contains data for two characters that can have two states (`A` and `B` for the first one, and `0` and `1` for the second one). Let's examine the file line by line:

- Line 1: NEXUS file header.
- Lines 3, 9, 11 and 17: these lines mark the start ("`Begin Rates;`") and end ("`End;`") of the rate blocks.
- Lines 4 and 12: these lines specify to which character each block refers to.
- Lines 5, 8, 13 and 16: these lines mark the start and end of the rate specification.
- Lines 6, 7, 14 and 15: these lines specify that every rate should be estimated using Bayesian sampling, and provide the prior for each one.

2.3.3 Pis

A pi block specifies the root node priors.

An example of a NEXUS file containing a pi block is the following:

```
1 #NEXUS
2
3 Begin Pi;
4     Character: 0;
5     Default: Dirichlet(1);
6 End;
```

Let's examine the file line by line:

- Line 1: NEXUS file header.
- Lines 3 and 6: these lines mark the start ("`Begin Pi;`") and end ("`End;`") of the pi block.
- Line 4: this line specifies to which character the block refers to.
- Line 5: this line specifies that all root node priors should be estimated using Bayesian sampling, and that a flat Dirichlet prior should be used.

3 Program installation

For installation and compilation instructions, please see the GitHub repository for sMap: https://github.com/arklumpus/sMap.

4 Using sMap

4.1 Running sMap

sMap is a command-line program, therefore it must be run from the command line (on any operating system).

4.1.1 Windows 10, Windows 8.1, Windows 8, Windows 7

- Press Windows+R to open the "Run command" window
- Type in `cmd` and press enter: the Windows command prompt will open
- (Optional) The current active folder is shown at the left, before the `>` symbol; to move to the folder with the files you want to analyse, type `cd ` (note the space after `cd`) and drag-and-drop the folder containing the files into the command prompt window (the path to that folder will appear). Press enter.
- If you have added the sMap executable folder to your PATH, you can simply type `sMap` and press enter to run the program (you will get an error message and a couple of "Usage" lines). If you get an error along the lines of `'sMap' is not recognized as an internal`

`or external command, operable program or batch file`, then you might not have added the proper folder to your PATH

- If you haven't added the sMap executable folder to your PATH, you will have to type the full path to `sMap.exe` (*hint*: you can drag-and-drop the executable into the command prompt window to have it appear automatically). If you press enter, you will receive the error message and "Usage" lines mentioned above

4.1.2 Linux, macOS and other operating systems

- Open a console terminal window using the tools provided by your operating system
- (Optional) `cd` to the folder containing the files you need to analyse
- If you have created symlinks as mentioned in the installation instructions, you can just type `sMap` and press enter to run the program (you will get an error message and a couple of "Usage" lines).
- If you haven't created the symlinks, you will need to type the path to the sMap executable (e.g.: `/path/to/sMap`). If you press enter, you should still get the error message and couple of "Usage" lines mentioned above.

5 Tutorials

This section contains a list of tutorials aimed at enabling users to quickly gain confidence with the options of sMap. For a detailed description of the analyses performed in each tutorial, please see the relevant section of this document (which is usually also mentioned in the tutorial).

These tutorials use datasets that have been analysed in Huelsenbeck, Nielsen and Bollback, 2003 (which were themselves already published datasets): a dataset on Cerataphidini aphids (Stern, 1998) and one on Pontederiaceae plants (Kohn *et al.*, 1996; Graham *et al.*, 1998).

The tutorials also assume that you installed sMap in such a way that you can recall it from any location (i.e. that you added the sMap folder to the PATH variable, or that you created the appropriate symlinks). If you haven't, you will have to specify the path to the sMap executable every time.

5.1 Dataset description

A detailed description of the datasets used is beyond the scope of these tutorials, but a brief overview is given below.

5.1.1 Cerataphidini dataset

This dataset originates from the phylogenetic analysis of aphids of the tribe Cerataphidini by Stern (1998). For these tutorials, the original cytochrome oxydase subunit I and II sequences were re-aligned with MAFFT v7.271 (Katoh and Standley, 2013) with parameters `--maxiterate 1000 --localpair`.

Positions with a proportion of gaps greater than 85% were deleted from the alignment, and the cleaned alignments were used with MrBayes v3.2.6 (Ronquist and Huelsenbeck, 2003) to run a strict clock analysis under a GTR+G+I model, with a birth-death prior on branch lengths and with *Neothoracaphis yanonis* constrained as the outgroup (other settings were left at their default values).

A 50% majority-rule consensus tree as well as a bifurcating tree with all compatible groups were generated; 1'000 tree samples were collected from the chain files. All trees were converted to NEWICK format.

The character being analysed for this dataset is the presence of "horned soldiers" in the aphid species. Data for this character were extracted from Huelsenbeck, Nielsen and Bollback, 2003.

5.1.2 Pontederiaceae dataset

This dataset originates from the phylogenetic analysis of plants of the family Pontederiaceae by Kohn *et al.* (1996) and Graham *et al.* (1998). For these tutorials, the original *rbcL* an *ndhF* sequences were re-aligned with MAFFT v7.271 (Katoh and Standley, 2013) with parameters `--maxiterate 1000 --localpair`.

Positions with a proportion of gaps greater than 85% were deleted from the alignment, and the cleaned alignments were used with MrBayes v3.2.6 (Ronquist and Huelsenbeck, 2003) to run a strict clock analysis under a GTR+G+I model, with a birth-death prior on branch lengths and with *Philydrum_lanuginosum* constrained as the outgroup (other settings were left at their default values).

A 50% majority-rule consensus tree as well as a bifurcating tree with all compatible groups were generated; 1'000 tree samples were collected from the chain files. All trees were converted to NEWICK format. The outgroup *Philydrum_lanuginosum* was then removed from all the trees.

The characters being analysed for this dataset are flower morphology and self-incompatibility. Data for these characters were extracted from Huelsenbeck, Nielsen and Bollback, 2003.

5.2 Running a ML analysis

In this quick tutorial, we will run the simplest possible analysis of the Cerataphidini dataset, under the default settings.

You can download the files for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial1/Tutorial1.zip. Extract the compressed folder in a suitable location.

When no model specification information is provided, sMap runs the analysis under an ARD model (see 1.2.2.1) whose parameters are specified using Maximum-Likelihood (ML), with equal root node priors.

The only data we are going to provide to the program are: the data about the character states at the tips (contained in the file `Cerataphidini.txt`), and the bifurcating phylogenetic tree of these species (contained in the file `Cerataphidini.bi.tre`). You may want to inspect the data file with a text editor, and the tree file with a tree viewer program such as FigTree (http://tree.bio.ed.ac.uk/software/figtree/). Please note, in particular, the low support values for some of the branches of the tree.

To go on with this tutorial:

- Open a command line window and move to the folder where you have downloaded the two files.
- Create a new folder to contain the results of the analysis; name it, e.g., **`Aphids_ML_ARD_bi`** to reflect the analysis we are doing (you can choose a different name, but be sure to change it in the commands that follow). You can do this using the graphical interface of your operating system's file browser, or in the command line by typing `mkdir` **`Aphids_ML_ARD_bi`** and pressing enter.
- Run the sMap analysis by typing: **`sMap`** `-t Cerataphidini.tre -d Cerataphidini.txt -o` **`Aphids_ML_ARD_bi`**`/tutorial1 -n 1000` and pressing enter. Here is a description of the line you just entered (for the full description of all possible command-line parameters, see 6.1.5):
    - **`sMap`**: invokes the program. If you haven't added the sMap folder to the PATH variable, or you haven't created a symlink, you will have to replace this with the full path to the sMap executable (see 4.1)

    - `-t Cerataphidini.tre`: specifies that the simulations must be run under the trees in this file. Since the file only contains one tree, the same tree will be also used to summarise the results and estimate ML parameters
    - `-d Cerataphidini.txt`: specifies that the character state data is contained in this file
    - `-o` **`Aphids_ML_ARD_bi`**`/tutorial1`: specifies the output prefix. All files created by the program will be called `tutorial1.something`, and they will be found in the **`Aphids_ML_ARD_bi`** subfolder
    - `-n 1000`: specifies the number of simulations to run
- The program will start and perform a ML optimization of the model parameters. Once this is finished, it will print the ML, AIC, AICc and BIC (which will be useful in model selection, see 1.2.3), as well as the ML parameter estimates. It will then start computing priors and posteriors and simulating the character state histories. Wait for the program to finish and exit.

Let's keep in mind these values:

ln-ML	-11.82
AIC	27.64
AICc	28.03
BIC	30.69
r(A→P)	0.222
r(P→A)	2.388

The program will create many different files in the output folder. Here is a brief description of them:

File	Description
`tutorial1.mean.params.txt`	Contains the mean parameter estimates for the model. When parameters are estimated by ML, these are the same as the ML estimates.
`tutorial1.paramNames.txt`	Contains the names of the parameters that need to be estimated. This will be useful in cases where you want to supply precomputed values for these parameters.
`tutorial1.params.txt`	Contains the values of the model parameters, as sampled for each tree. When parameters are estimated by ML, all of the values in each column are identical.
`tutorial1.priors.log`	Contains the node state posterior-predictive samples for each node of each sampled tree.
`tutorial1.posteriors.log`	Contains the node state posterior samples for each node of each sampled tree.
`tutorial1.loglikelihoods.log`	Contains the log-likelihoods for each state at each node of each sampled tree
`tutorial1.prior.mean.pdf`	Contains a plot of the summary tree, in which each node has a pie chart that represents the node state prior for that node, averaged over all the sampled trees.
`tutorial1.likelihood.mean.pdf`	Contains a plot of the summary tree, in which each node has a "target" chart that represents the node state likelihoods for that node, averaged over all the sampled trees.

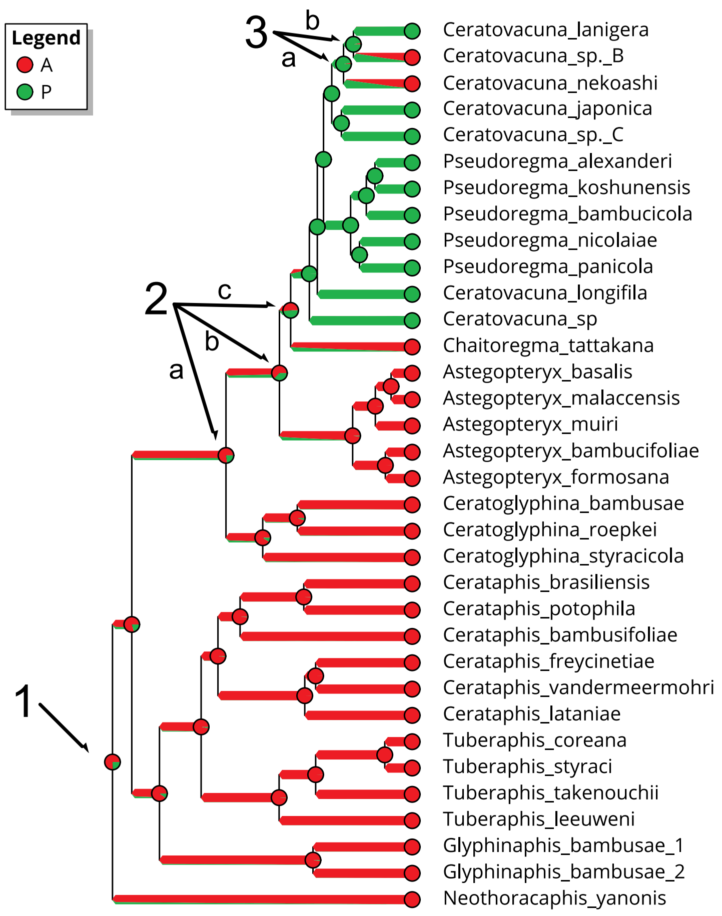
`tutorial1.posterior.mean.pdf`	Contains a plot of the summary tree, in which each node has a pie chart that represents the node state posterior for that node, averaged over all the sampled trees.
`tutorial1.all.mean.pdf`	Contains a plot of the summary tree, in which each node has a chart representing priors on the vertical axis, likelihoods on the horizontal axis, and posteriors as the (relative) surface area of the coloured rectangles.
`tutorial1.smap.tre`	Contains the simulated histories, one per line, in a NEWICK-derived format (the same used by the `write.simmap` function of R package `phytools` (Revell, 2012), hence this file can be read by the `read.simmap` function of the same package).
`tutorial1.smap.bin`	Contains the simulated histories (and some additional information) in a binary format that can be opened by other programs included in sMap.
`tutorial1.smap.pdf`	Contains the summary of the stochastic mapping analysis: a plot of the summary tree, where each node has a pie chart representing the node state posteriors (these can be different by those shown in the posterior mean file, see 6.7.1.1) and each branch is coloured according to the posterior probability of each state at each point in time of that branch
`tutorial1.branchprobs.txt`	For each branch of each sampled tree, contains the posterior-predictive probability of that branch. This is the probability, given the sampled state for the start and end points of the branch, that a stochastic simulation using the current fitted model will be coherent with the sampled states. This is a measure of how “difficult” the history sampling step will be (i.e. how many attempts will have to be made for each branch), as well as of how “reasonable” the observed data is, given the model.
`tutorial1.branchprobs.pdf`	Contains a summary plot of the posterior-predictive probability of each branch. For each branch, the average probability, as well as the range in probabilities across all the sampled trees is shown.
`tutorial1.branchprobs.tree.pdf`	Contains a plot of the average and minimum branch posterior probability for each branch (plotted on the summary tree).

The file we are most interested with is the last one, `tutorial1.smap.pdf`. If you open it with your favourite PDF viewer, you will see something similar to the figure on the right (except for the arrows, which have been added for ease of discussion).

The figure contains a plot of the tree we supplied the program, and each node has a pie chart, which represents the posterior probability that the node was in each state. Each branch also has a coloured plot, which is coloured proportionally to the probability of the branch being in each state at each point in time.

Interesting features are:

1. Even though most of the basal species of aphids do not have a horned soldier, according to this model there is a non-negligible probability that the ancestor of all Cerataphidini had a horned soldier
2. There is quite a high probability the horned soldiers evolved before the last common ancestor of all the modern species that have a horned soldier
3. In the evolution of the *Ceratovacuna* genus, horned soldiers have most likely been lost repeatedly



We will make sense of these points in the following tutorials; for now, let us collect some quantitative measures about these nodes. To do this, we will use the NodeInfo utility.

- In the command line, move to the folder with the results of the analysis: `cd` **`Aphids_ML_ARD_bi`**
- To obtain the posterior probabilities for each state for the root node (1 in the figure), type **`NodeInfo`** `-s tutorial1.smap.bin -m Neothoracaphis_yanonis Ceratovacuna_lanigera --batch` and press enter. Here is a description of the line you just entered (for the full description of all the possible command-line parameters, see 6.3.3):
    - **`NodeInfo`**: invokes the program. If you haven't added the sMap folder to the PATH variable, or you haven't created a symlink, you will have to replace this with the full path to the NodeInfo executable (see 4.1)
    - `-s tutorial1.smap.bin`: specifies the input file, which should be the `smap.bin` file produced by the sMap run
    - `-m Neothoracaphis_yanonis Ceratovacuna_lanigera`: specifies the node we are interested in, as the least inclusive monophyletic node that contains both *Neothoracapis yanonis* and *Ceratovacuna lanigera*. If you are in a Linux or MacOs environment and have enabled tab-completion for sMap, you should be able to use it to complete the taxon names.
    - `--batch`: specifies that the program should only print the state probabilities for the selected node(s) and exit.
- The output of the program should be something like this:

```
0
A          0.7590
P          0.2410
```

Where:

- `0` is the identification number of the node ("node id")

  - `A          0.7590` means that the posterior probability of the node being in state A is 75.9%
  - `P          0.2410` means that the posterior probability of the node being in state P is 24.1%
- To obtain the posterior probabilities for node 2a in the figure, use **`NodeInfo`** `-s tutorial1.smap.bin -m Ceratoglyphina_styracicola Ceratovacuna_lanigera --batch`. You should get a similar output as before.
- Repeat this to obtain the posterior probabilities for nodes 2b, 2c, 3a and 3b, changing the names of the taxa to specify the node you are interested in. In the end, you should get results similar to these (the node ids should be exactly the same, the probabilities may vary slightly):

Node (in figure)	Node id	$\mathbb{P}(A)$	$\mathbb{P}(P)$
1	0	75.9%	24.1%
2a	2	77.5%	22.5%
2b	3	63.0%	37.0%
2c	4	51.8%	48.2%
3a	9	2.8%	97.2%
3b	10	3.1%	96.9%

This is the end of this tutorial. In the next tutorial we will analyse the same dataset using a different model.

5.3 Running a ML analysis with an ER model

In this tutorial we will analyse again the Cerataphidini dataset, still under ML, using an equal-rates (ER) model, which assumes that the rates of change are all equal (see 1.2.2.2).

To do this, we need to supply sMap a model specification file, which is a kind of NEXUS file (Maddison, Swofford and Maddison, 1997) including a "Dependency" and/or "Rates" and/or "Pi" block. You can download the model file for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial2/Cerataphidini.model.ML.ER.nex. Save the file in the same folder as the data file and tree file of the previous tutorial.

First, let's open the model file (`Cerataphidini.model.ML.ER.nex`) using a text editor and look at it:

```
#NEXUS

Begin Rates;

    Character: 0;

    Rates:
        A > P: ML
        P > A: Equal(A > P)
        ;

End;
```

Let's examine it line by line.

- 1: This is the NEXUS file format header. It should be present in all NEXUS files.
- 3, 12: These lines mark the start and end of the “Rates” block.
- 5: This line specifies which character (in a 0-based index) the block refers to. In this case we only have one character, so this line can be omitted, but it’s good practice to always include it.
- 7: This line marks the beginning of the actual rate specification.
- 8: `A > P` is the rate of going from state A to state P. Here we are specifying that it should be estimated using maximum likelihood (`ML`).
- 9: P > A is the rate of going from state P to state A. Here we are specifying that it should be equal to the opposite rate (`Equal(A > P)`). This specifies an equal-rates model. If we had instead used `ML` to estimate it using maximum likelihood, we would have had the same ARD model as the previous tutorial.
- 10: The semicolon (`;`) marks the end of the rate specification.

When we supply this file to sMap, the A > P rate will be estimated using maximum likelihood, while the P > A rate will be constrained to be equal to that. It would have been the same to estimate the P > A rate and constrain the A > P rate to be equal to this.

Now:

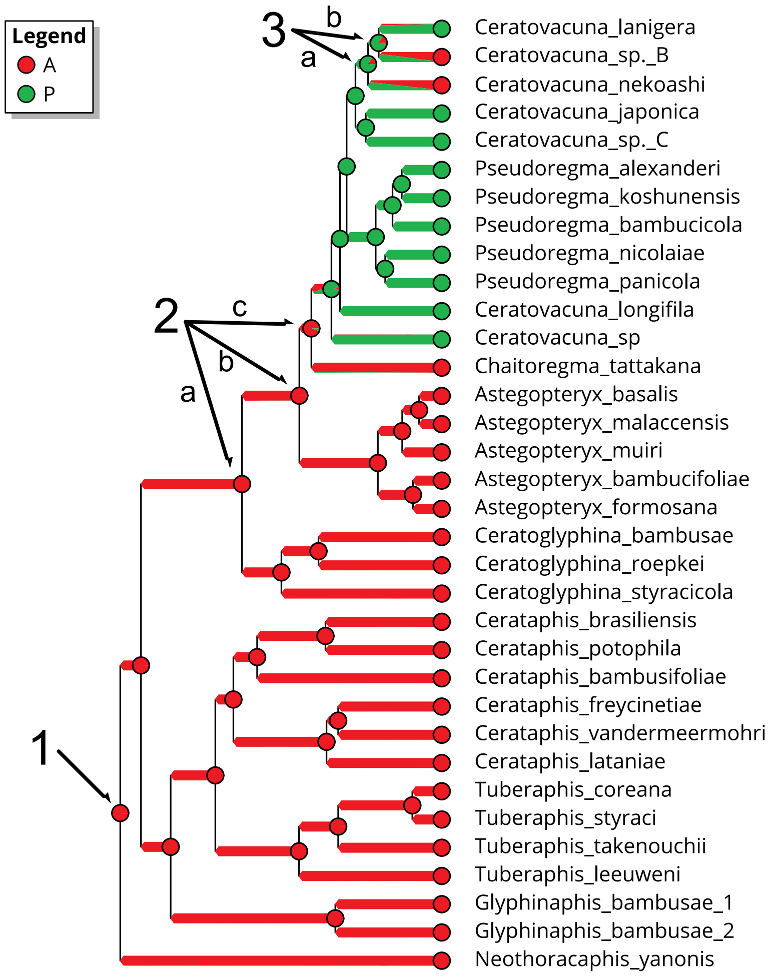
- Create a new folder named **`Aphids_ML_ER_bi`** to host the results of the analysis.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.tre -d Cerataphidini.txt -o Aphids_ML_ER_bi/tutorial2 -n 1000 -i Cerataphidini.model.ML.ER.nex` and pressing enter. The main difference between this and the command in the previous tutorial is:
  - `-i Cerataphidini.model.ML.ER.nex`: specifies a model input file, which may contain specifications about the rates, root node priors and character relationships.
- The program will estimate the maximum-likelihood rate and report ML, AIC, AICc and BIC values.

Once again, let’s keep track of them:

ln-ML	-13.62
AIC	29.25
AICc	29.37
BIC	30.77
r(A→P)	0.487
r(P→A)	0.487

Looking at the tutorial2.smap.pdf output file, the posterior probabilities for some of the nodes have changed. To obtain the new values for them:

- In the command line, move to the folder with the results of the analysis: `cd Aphids_ML_ER_bi`
- Type `NodeInfo -s tutorial2.smap.bin -n 0,2,3,4,9,10 --batch` and press enter. The main difference between this line and the one in the previous tutorial is:
  - `-n 0,2,3,4,9,10`: this specifies the nodes we are interested in by their node id. This way, we can obtain information about all of the nodes in a single command.
- The output of this command will be similar to the one in the previous tutorial, except for the fact that it will contain an entry for each node that we have specified.



The values should be similar to these:

Node (in figure)	Node id	$\mathbb{P}(A)$	$\mathbb{P}(P)$
1	0	99.1%	0.09%
2a	2	100%	0%
2b	3	98.6%	1.4%
2c	4	94.7%	5.3%
3a	9	16.2%	83.8%
3b	10	16.4%	83.6%

The results between the ER model and the ARD model are quite different: which one should we use? By looking at the AIC(c) and BIC values, we can see that they are quite close, thus it's not an easy choice. In the next tutorials we will run the analyses in a Bayesian framework, and then we will compute marginal likelihoods to more accurately compare the models.

5.4 Running a Bayesian analysis with an ARD model

In this tutorial we will analyse again the Cerataphidini dataset, using an ARD model and a Bayesian approach.

To do this, we will use a model file that specifies a prior distribution for the two transition rates and for the root node prior. We will use a flat Dirichlet prior for the root node priors and, since we do not have any prior belief about the distribution of the rates, we will use an "empirical Bayes" approach and use as priors log-normal distributions centred around the maximum likelihood estimate for each gene. For a list of all the possible prior distributions, see 6.1.1.2.1.

You can download the files for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial3/Tutorial3.zip. Extract them and put them in the same folder as the tree file and data file from the first tutorial. We will use the `Cerataphidini.treedist` file later, for now let's examine the model file `Cerataphidini.model.Bayes.ARD.nex`:

```
 1 #NEXUS
 2
 3 Begin Pi;
 4
 5     Character: 0;
 6
 7     Default: Dirichlet(1);
 8
 9 End;
10
11 Begin Rates;
12
13     Character: 0;
14
15     Rates:
16         A > P: LogNormal(-1.50507789710986, 1)
17         P > A: LogNormal(0.870456195530356, 1)
18         ;
19
20 End;
```

These are the main features of this file, line by line:

- 3, 7: start and end of the "Pi" block, which specifies the root node prior.
- 7: "Default" value for the root node prior. This value is used for those states that are not explicitly assigned another value. In this case, the root node prior for both states A and P will have a flat Dirichlet distribution. For more details, see 2.3.3.
- 16, 17: priors for the rates. The LogNormal distribution takes two parameters: the first is the mean of the distribution's logarithm (in this case, the logarithm of the estimated values from the ML analysis in the first tutorial) and the second is the standard deviation of the distribution's logarithm (in this case, `1`, in order to sample slightly more than an order of magnitude of transition rates).

Let's run the analysis:

- Create a new folder named **`Aphids_Bayes_ARD_bi`** to host the results of the analysis.
- Run the sMap analysis by typing: **`sMap`** `-t Cerataphidini.tre -d Cerataphidini.txt -o` **`Aphids_Bayes_ARD_bi`**`/tutorial3 -n 1000 -i`

`Cerataphidini.model.Bayes.ARD.nex` and pressing enter. This is similar to the command used in the previous tutorial.

- The program will estimate the step sizes to use in the MCMC sampling, and then start the MCMC sampling, which will run until the convergence criteria have been reached (see 6.1.1.2.2). For this dataset, this shouldn't take too long.

In the output folder there will be some new files that were not produced during a ML analysis:

File	Description
`tutorial3.params.pdf`	Contains plots of the probability distributions of the sampled parameters, showing the posterior distributions as histograms and the prior distributions as curves.
`tutorial3.run1.log` `tutorial3.run2.log`	Contain the values that were sampled during the MCMC runs. There will be one file for each run (by default, two parallel runs are used). You can inspect these files using e.g. Tracer (Rambaut *et al.*, 2018) or the ChainMonitor utility (see 6.2) to verify that the chains have reached convergence.

The tutorial3.smap.pdf file should show similar results to the first analysis.

In a Bayesian analysis, it is possible to supply to sMap, instead of a single tree, a posterior distribution of trees (which can have different topologies and branch lengths), in order to take into account uncertainties in the phylogeny. The `Cerataphidini.treedist` file contains a list of 1000 clock-like trees, which we will now use to re-run the analysis:

- Run the sMap analysis by typing: `sMap -t Cerataphidini.treedist -T Cerataphidini.tre -d Cerataphidini.txt -o Aphids_Bayes_ARD_bi/tutorial3 -n 1000 -i Cerataphidini.model.Bayes.ARD.nex` and pressing enter. The main differences between this command and the previous one are:
    - `-t Cerataphidini.treedist`: this option now points to the file containing the list of trees.
    - `-T Cerataphidini.tre`: when more than one tree is used to perform the analysis, it is necessary to also provide a tree that will be used to summarise the results (this could be e.g. a consensus tree).
- Note that this will overwrite the previous analysis.

To summarise the results of an analysis that uses multiple trees, sMap will look at each branch at each point in time and determine the state of that branch in that moment by comparing all the simulation in which the branch exists at the particular point in time.

A new file will be produced in the output folder:

File	Description
`tutorial3.ssize.pdf`	Contains a plot showing the sample size for each branch at each point in time (i.e. the proportion of simulations in which the branch exists at the particular point in time). Thicker branches mean higher sample sizes.

If you look at the `tutorial3.smap.pdf` file, you should notice that it is a bit different from the files produced in both the first and second tutorial.

Once again, let's get the posterior probability values for the nodes we are interested in:

- In the command line, move to the folder with the results of the analysis: `cd Aphids_Bayes_ARD_bi`
- Type `NodeInfo -s tutorial3.smap.bin -n 0,2,3,4,9,10 --batch` and press enter. This is the same command that we used in the previous tutorial.

The values should be similar to these:

Node (in figure)	Node id	$\mathbb{P}(A)$	$\mathbb{P}(P)$
1	0	74.20%	25.80%
2a	2	80.89%	19.11%
2b	3	72.73%	27.27%
2c	4	59.32%	40.68%
3a	9	0.68%	99.32%
3b	10	1.79%	98.21%

This is the end of this tutorial. In the next tutorial, we will run a similar Bayesian analysis, using an ER model.

5.5 Running a Bayesian analysis with an ER model

In this tutorial we will analyse the Cerataphidini dataset with an ER model and a Bayesian approach.

In the model file we will thus specify the prior distribution for the root node prior and for only one of the transition rates, and we will set the two rates to be equal.

You can download the file for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial4/Cerataphidini.model.Bayes.ER.nex. Put it in the tutorial folder. If you open it in a text editor, you will see its contents:

```
#NEXUS

Begin Pi;

    Character: 0;

    Default: Dirichlet(1);

End;

Begin Rates;

    Character: 0;

    Rates:
        A > P: LogNormal(-0.718590611566901, 1)
        P > A: Equal(A > P)
        ;

End;
```

The only differences with the file from the previous tutorial are in line 16 (we used the logarithm of the rate estimate from the ML analysis with an ER model) and 17 (we specify that the $P \rightarrow A$ rate should be equal to the $A \rightarrow P$ rate).

Let's run the analysis:

- Create a new folder named `Aphids_Bayes_ER_bi` to host the results of the analysis.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.treedist -T Cerataphidini.tre -d Cerataphidini.txt -o Aphids_Bayes_ER_bi/tutorial4 -n 1000 -i Cerataphidini.model.Bayes.ER.nex` and pressing enter. This is similar to the command used in the previous tutorial; note that we are supplying the posterior distribution of trees.
- The program will estimate the step sizes to use in the MCMC sampling, and then start the MCMC sampling, which will run until the convergence criteria have been reached (see 6.1.1.2.2). Again, for this dataset, this shouldn't take too long.

The output folder will contain the usual files; the tutorial4.smap.pdf will again show slightly different results from the other tutorials. Let's get the posterior probabilities for the nodes we're interested in:

- In the command line, move to the folder with the results of the analysis: `cd Aphids_Bayes_ER_bi`
- Type `NodeInfo -s tutorial4.smap.bin -n 0,2,3,4,9,10 --batch` and press enter. This is the same command that we used in the previous tutorials.

The values should be similar to these:

Node (in figure)	Node id	$\mathbb{P}(A)$	$\mathbb{P}(P)$
1	0	98.90%	1.10%
2a	2	99.58%	0.42%
2b	3	97.95%	2.05%
2c	4	89.47%	10.53%
3a	9	3.88%	96.12%
3b	10	4.94%	95.06%

This is the end of this tutorial. In the next tutorial we will look at how to choose between different models (in this case, between the ER and ARD models, which give different results).

5.6 Bayesian model selection with marginal likelihoods

The problem of model selection is widespread in statistical analysis, and multiple model selection criteria have been developed. When a maximum-likelihood estimate is available, it is usually possible to compute values for the (corrected) Akaike information criterion (AIC/AICc) (Akaike, 1998) or the Bayesian information criterion (BIC) (Schwarz, 1978) and then choose the model with the lowest value.

However, in some cases, the values are very close, and it is thus not possible to assume that a model is a clear "winner" over another. For example, compare AIC/AICc/BIC values for the ARD model obtained in 5.2 (27.64/28.03/30.69) against those for the ER model obtained in 5.3 (29.25/29.37/30.77). In these cases, the approach is usually to compute appropriate weights and then "average" the results obtained with the different models according to those weights.

We can add a Bayesian twist to this by considering marginal likelihoods (see 1.2.3.3), which can be computed by sMap using the stepping-stone algorithm (Xie *et al.*, 2011). In this tutorial, we will compute marginal likelihoods for the ARD and ER models for the Cerataphidini dataset and use them to compute model posterior probabilities which can be used to appropriately mix the two models.

The first step is to compute the marginal likelihood for each model that we want to compare. To do this, assuming that you already have the files for tutorials 5.4 and 5.5, move to the folder that contains them and:

- Create a new folder named **`Aphids_Bayes_ARD_ss_bi`** to host the results of the stepping-stone (ss) analysis.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.treedist -T Cerataphidini.tre -d Cerataphidini.txt -o Aphids_Bayes_ARD_ss_bi/tutorial5 -n 1000 -i Cerataphidini.model.Bayes.ARD.nex -ss` and pressing enter. This is similar to the command used in the previous tutorial; note that we are supplying the posterior distribution of trees. The main difference is the `-ss` option: this enables the stepping-stone algorithm and computes marginal likelihoods.
- The program will run a standard MCMC sampling, followed by the stepping-stone analysis; each step will run until the convergence criteria have been reached (see 6.1.1.2.2)[4]. This will be slower than a normal Bayesian analysis (it will take around 10x longer).
- The (log-)marginal likelihood value for the ARD model will be displayed at the end of the analysis: take note of this (it should be something around -12.60).
- Perform the same analysis for the ER model: create a new folder named **`Aphids_Bayes_ER_ss_bi`**.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.treedist -T Cerataphidini.tre -d Cerataphidini.txt -o Aphids_Bayes_ER_ss_bi/tutorial5 -n 1000 -i Cerataphidini.model.Bayes.ER.nex -ss` and pressing enter.
- Take note of the log-marginal likelihood (it should be around -13.97).

A new file will be produced in each output folder:

File	Description
`tutorial5.marginal.likelihoods.txt`	Contains a summary of the stepping-stone analysis, detailing the contribution to the marginal likelihood of each step. The marginal likelihoods are computed for each set of independent characters, and the overall marginal likelihood for the whole data is also reported.

We can now use the marginal likelihoods to compute the posterior probability of each model (see 1.2.3.3). The posterior probability for a model is:

[4] Sometimes, the convergence criteria used for a standard analysis won't be efficient for a stepping-stone analysis (i.e. they will fail to recognise that the run has converged, even though it has); in a real-world setting, you would periodically inspect the chains and then trigger a manual interrupt when you feel that convergence has been reached (see 6.1.1.2.2), but for the purpose of speeding up this analysis, you can add the `--max-cov=1` switch, which will effectively disable the CoV criteria and only assess convergence by the number of samples and ESS (see 6.1.1.2.2).

$$\mathbb{P}(\text{Model} \mid \text{Data}) = \frac{\mathbb{P}(\text{Model}) \cdot \mathbb{P}(\text{Data} \mid \text{Model})}{\sum_{i=1}^{n} \mathbb{P}(\text{Model}_\text{i}) \cdot \mathbb{P}(\text{Data} \mid \text{Model}_\text{i})}$$

Where $\mathbb{P}(\text{Model})$ is the prior probability for the model and $\mathbb{P}(\text{Data} \mid \text{Model})$ is the marginal likelihood for that model. If we use a "quasi-uniform" prior for the models (i.e. we assume that some models are impossible and have prior probability 0, while all the others have the same prior probability), this simplifies to:

$$\mathbb{P}(\text{Model} \mid \text{Data}) = \frac{\mathbb{P}(\text{Data} \mid \text{Model})}{\sum_{i=1}^{n} \mathbb{P}(\text{Data} \mid \text{Model}_\text{i})}$$

In our case, we have two models (ARD and ER), and we know the logarithm for the marginal likelihood of each. Thus:

$$\mathbb{P}(\text{ARD} \mid \text{Data}) = \frac{\mathbb{P}(\text{Data} \mid \text{ARD})}{\mathbb{P}(\text{Data} \mid \text{ARD}) + \mathbb{P}(\text{Data} \mid \text{ER})} = \frac{e^{-12.60}}{e^{-12.60} + e^{-13.97}} \approx 79.74\%$$

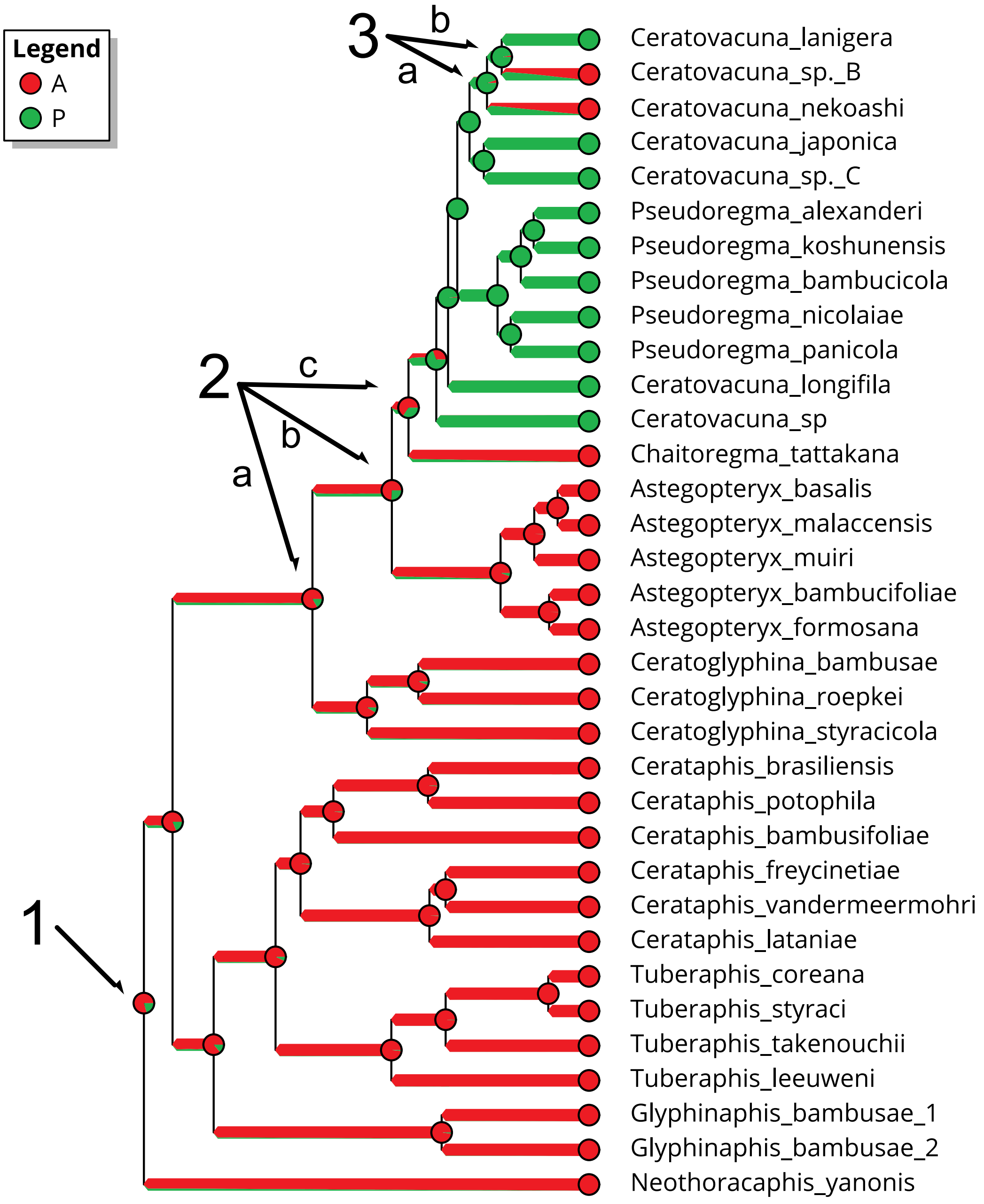
And, similarly:

$$\mathbb{P}(\text{ER} \mid \text{Data}) = \frac{\mathbb{P}(\text{Data} \mid \text{ER})}{\mathbb{P}(\text{Data} \mid \text{ARD}) + \mathbb{P}(\text{Data} \mid \text{ER})} = \frac{e^{-13.97}}{e^{-12.60} + e^{-13.97}} \approx 20.26\%$$

We can now use these posterior probabilities to "blend" the stochastic maps that we have already obtained (each of which is *conditioned* on one of the two models). To do this, we use the Blend-sMap utility.

- First, create a new folder named **`Aphids_Bayes_blended_bi`** to host the results of the analysis.
- To blend the previous analyses using Blend-sMap, type: `Blend-sMap -o Aphids_Bayes_blended_bi/tutorial5.blended.smap.bin -n 5000 -s Aphids_Bayes_ARD_ss_bi/tutorial5.smap.bin,79.74 -s Aphids_Bayes_ER_ss_bi/tutorial5.smap.bin,20.26` and press enter. Here is a description of the command-line parameters (see 6.4 for a full description of the available options):
    - `-o Aphids_Bayes_blended_bi/tutorial5.blended.smap.bin`: this parameter specifies the name of the output file.
    - `-n 5000`: this parameter specifies the number of histories in the output file. Here we have chosen 5000 so that the least probable model, which will account for about 20% of these simulations, will be represented by around 1000 histories. There isn't a hard rule to choose this value, but it shouldn't make too much of a difference usually. When the number of required simulations is higher than the number of simulations in the input files, they will be randomly resampled.
    - `-s Aphids_Bayes_ARD_ss_bi/tutorial5.smap.bin,79.74` and `-s Aphids_Bayes_ER_ss_bi/tutorial5.smap.bin,20.26`: these specify the input files. The numbers after the comma are the weights of each file. The -s option can be specified multiple times (here it is given twice), and the weights can be expressed in arbitrary units (they will be normalised by the program).
- The Blend-sMap program does not produce a PDF plot by default, thus we will create one with the Plot-sMap utility. This utility will be the focus of a future tutorial (see 5.8), so for now we can just use the following command, which only specifies the sMap file and tells the program to create a plot with default settings and exit: `Plot-sMap -s Aphids_Bayes_blended_bi/tutorial5.blended.smap.bin -batch`. This will create a file called `tutorial5.blended.smap.pdf` in the same folder as the sMap file.

If you open this PDF file you will see the results of the average between the ARD and ER analyses:



We can now obtain the posterior probabilities for the nodes of interest, using the NodeInfo utility:

- In the command line, move to the folder with the results of the analysis: `cd` **`Aphids_Bayes_blended_bi`**
- Type **`NodeInfo`** `-s tutorial5.blended.smap.bin -n 0,2,3,4,9,10 --batch` and press enter. This is the same command that we used in the previous tutorials.

The values should be similar to these:

Node (in figure)	Node id	$\mathbb{P}(A)$	$\mathbb{P}(P)$
1	0	79.60%	20.40%
2a	2	83.49%	16.51%

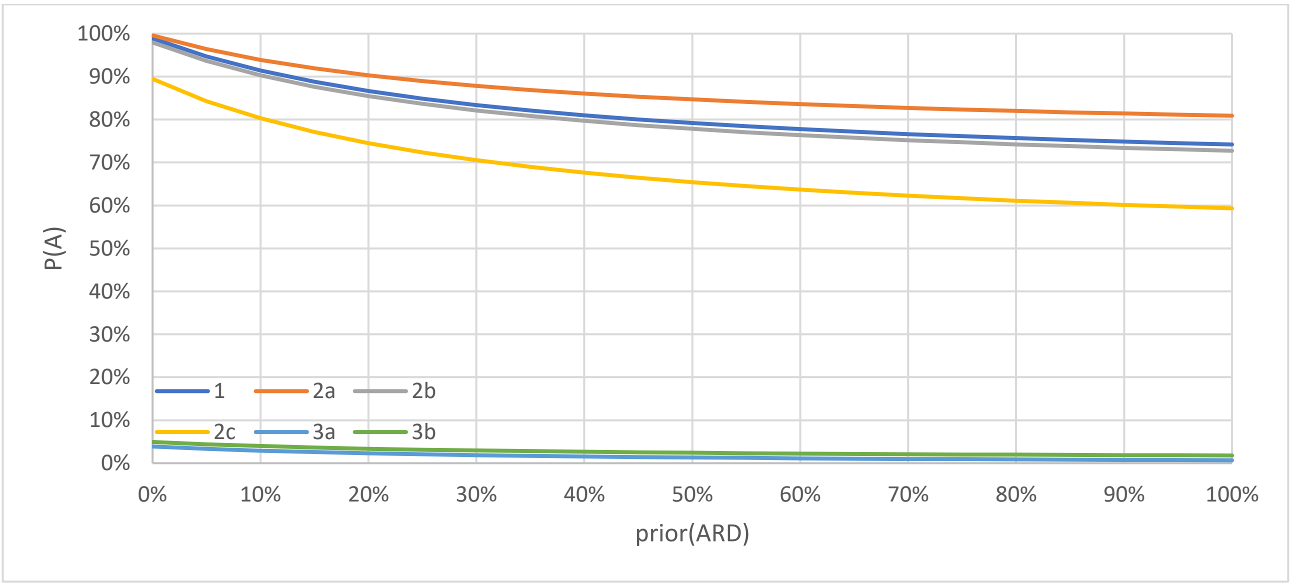
2b	3	74.30%	25.70%
2c	4	65.63%	34.37%
3a	9	4.79%	95.21%
3b	10	2.61%	97.39%

These are the "final" results of our analyses using the bifurcating tree of Cerataphidini; we can thus start to interpret them. First, we cannot exclude that the last common ancestor (LCA) of Cerataphidini possessed a horned soldier caste, even though this has a low probability of being true; similarly, we cannot say whether the ancestors in nodes 2a, 2b and 2c already had it. Finally, we can confidently say that the horned soldier was lost independently by *C. nekoashi* and *Ceratovacuna sp. B*, rather than having been lost by the ancestor in node 3a and then acquired again by *C. lanigera*.

This pattern of low support for many hypotheses is quite common for complex characters, which are the results of multiple interacting processes; a way to address this would be to investigate the mechanisms that allow a horned soldier caste to be present, and to analyse these using stochastic mapping. Furthermore, this phylogeny is not well-resolved (for example, internal relationships within the *Ceratovacuna* genus have especially low posterior probabilities, and a different position of *C. nekoashi* and *Ceratovacuna sp. B* would have a big effect on the stochastic mapping analysis): improving on the phylogenetics would also help address these uncertainties.

It is meaningful to note that the uncertainties (in the phylogeny and in the model selection) make the results of the stochastic mapping analysis *uncertain*, rather than *wrong*; this is the main advantage of a Bayesian approach, rather than a parsimony or maximum-likelihood based approach. In the following tutorial the uncertainty will be even more apparent, as we will use a 50% majority-rule consensus tree for the summarisation.

As a final note on this tutorial, let's consider the effect of the prior on the evolutionary model. In this tutorial, we assumed that both models had a 50% prior probability of being true; the following plot shows how the posterior probabilities for each node would change with different priors. You can download the Excel file used to produce this plot from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial5/PriorPlot.xlsx.



The horizontal axis shows the prior probability of the ARD model, while the vertical axis shows the resulting posterior probability for state A for each of the nodes of interest. These curves are reasonably flat (except maybe for node 2b), which means that the prior, in this case, does not have a huge effect on the results.

5.7 Analyses using a multifurcating summary tree

In this tutorial we will repeat the analyses of the previous tutorials (5.2, 5.3, 5.4, 5.5 and 5.6), using a multifurcating 50%-majority rule consensus tree as summary tree. The rationale behind this is that it is pointless to make inferences about nodes in the tree, when we don't even know whether those nodes exist or not: here we will instead only focus on nodes for which there is substantial evidence for their existence.

We will assume that you have already downloaded the files for the previous tutorials and have extracted them all in the same folder. You can find the file for this tutorial here https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial6/Cerataphidini.halfcompat.tre, download it and place it in the same folder as the other files. The `Cerataphidini.halfcompat.tre` file contains the multifurcating phylogenetic tree. The commands that we will use are the same as previous tutorials, but we will replace the tree contained in the Cerataphidini.tre file with the new file.

5.7.1 Run a ML analysis with an ARD model

- Create a new folder named **`Aphids_ML_ARD_multi`** to host the results of the analysis.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.halfcompat.tre -d Cerataphidini.txt -o Aphids_ML_ARD_multi/tutorial6 -n 1000` and pressing enter.
- If you want, look at the tutorial6.smap.pdf file in the output folder. Take note of the ML estimates for the rates: the $A \rightarrow P$ rate should be around 0.214, and the $P \rightarrow A$ rate should be around 1.646.

5.7.2 Run a ML analysis with an ER model

- Create a new folder named **`Aphids_ML_ER_multi`** to host the results of the analysis.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.halfcompat.tre -d Cerataphidini.txt -o Aphids_ML_ER_multi/tutorial6 -n 1000 -i Cerataphidini.model.ML.ER.nex` and pressing enter.
- If you want, look at the tutorial6.smap.pdf file in the output folder. Take note of the ML estimates for the rate: it should be around 0.454.

5.7.3 Run a Bayesian analysis with an ARD model, also estimating the marginal likelihood

- Create a new folder named **`Aphids_Bayes_ARD_ss_multi`** to host the results of the analysis.
- Edit the `Cerataphidini.model.Bayes.ARD.nex` file so that the prior distributions are centred on the new ML estimates for the rates. The prior for the $A \rightarrow P$ rate should be `LogNormal(-1.54177926396029, 1)`, while the prior for the $P \rightarrow A$ rate should be `LogNormal(0.498348102254878, 1)`.
- Run the sMap analysis by typing: `sMap -t Cerataphidini.treedist -T Cerataphidini.halfcompat.tre -d Cerataphidini.txt -o Aphids_Bayes_ARD_ss_multi/tutorial6 -n 1000 -i Cerataphidini.model.Bayes.ARD.nex -ss` and pressing enter (remember to add the `--max-cov=1` switch if the analysis fails to reach convergence with the default settings, as detailed in note 4 at page 29**Error! Bookmark not defined.**).
- If you want, look at the tutorial6.smap.pdf file in the output folder. Take note of the log-marginal likelihood estimate: it should be around -12.56.

5.7.4 Run a Bayesian analysis with an ER model, also estimating the marginal likelihood

- Create a new folder named **`Aphids_Bayes_ER_ss_multi`** to host the results of the analysis.
- Edit the `Cerataphidini.model.Bayes.ER.nex` file so that the prior distribution is centred on the new ML estimates for the rates. The prior should be `LogNormal -0.789658080940789, 1).`
- Run the sMap analysis by typing: **`sMap`** `-t Cerataphidini.treedist -T Cerataphidini.halfcompat.tre -d Cerataphidini.txt -o` **`Aphids_Bayes_ER_ss_multi`**`/tutorial6 -n 1000 -i Cerataphidini.model.Bayes.ER.nex -ss` and pressing enter (remember to add the `--max-cov=1` switch if the analysis fails to reach convergence with the default settings, as detailed in note 4 at page 29).
- If you want, look at the tutorial6.smap.pdf file in the output folder. Take note of the log-marginal likelihood estimate: it should be around -13.93.

5.7.5 Blend the previous analyses according to the model posterior probabilities

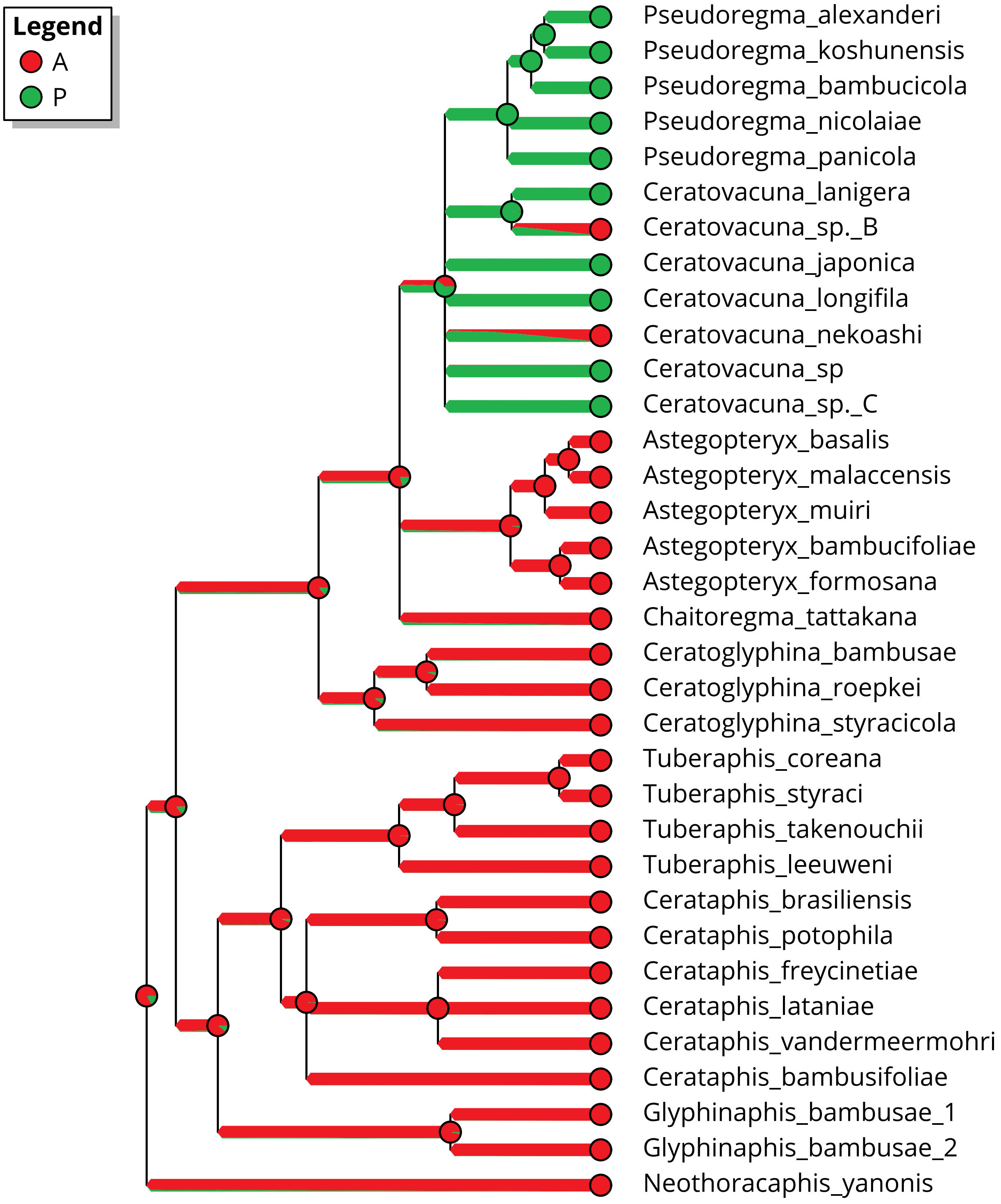
- Compute the model posterior probabilities for the ER and ARD models:

$$\mathbb{P}(\text{ARD}) = \frac{e^{-12.56}}{e^{-12.56} + e^{-13.93}} \approx 79.74\%$$

$$\mathbb{P}(\text{ER}) = \frac{e^{-13.93}}{e^{-12.56} + e^{-13.93}} \approx 20.26\%$$

- Create a new folder named **`Aphids_Bayes_blended_multi`** to host the results of the analysis.
- To blend the previous analyses using Blend-sMap, type: **`Blend-sMap`** `-o` **`Aphids_Bayes_blended_multi`**`/tutorial6.blended.smap.bin -n 5000 -s` **`Aphids_Bayes_ARD_ss_multi`**`/tutorial6.smap.bin,79.74 -s` **`Aphids_Bayes_ER_ss_multi`**`/tutorial6.smap.bin,20.26` and press enter.
- Plot the analysis using the Plot-sMap utility: type **`Plot-sMap`** `-s` **`Aphids_Bayes_blended_multi`**`/tutorial6.blended.smap.bin --batch` and press enter.

If you now look at the `tutorial6.blended.smap.pdf` file that has been produced (pictured below), you will see that the results are similar to the ones we already obtained by using the bifurcating tree, i.e. the only statement we can make with confidence is that the *Ceratovacuna sp. B* and *C. nekoashi* most likely lost the soldier caste independently, while we can't really be sure about the root of the tree, or about some of the internal nodes.



This is the end of this tutorial. In the next tutorial, we will use the Plot-sMap utility to make a better-looking plot of our stochastic mapping analyses.

5.8 Plotting a stochastic mapping analysis

When a sMap analysis is finished, the program creates a plot of the results using default settings. It is however usually advisable to replot the analysis, in order to make a display that reflects each different analysis. For example, large phylogenetic trees may be too compressed and illegible with the default settings, or the default colours may not be aesthetically pleasing.

The Plot-sMap utility is used to specify custom plot settings and should be enough to obtain a decent plot in most occasions. If you want to have even finer control over the plot elements, you should use a vector editing program such as Inkscape (http://www.inkscape.org) or Adobe Illustrator (https://www.adobe.com/it/products/illustrator.html) to manually edit the plot.

As all the other tools in the sMap suite, Plot-sMap is a command-line program. It can be useful to have a "real-time" view of the results of the plot, so that you can have an idea of the effect of the various options and change the values accordingly. To do this, you need to open the PDF plot produced by Plot-sMap using a PDF viewer that does not lock the file (unfortunately, if you open the PDF in Adobe Reader, it will prevent Plot-sMap from updating it until you close the file). On Windows, you can use for example the Sumatra PDF reader (https://www.sumatrapdfreader.org/free-pdf-reader.html), which will automatically update the PDF display when the file is updated. On Linux you can use e.g. Evince (http://projects.gnome.org/evince/), while on macOS you can use the included Preview program.

In this tutorial we will use Plot-sMap to plot the results of the analysis from tutorial 5.6. If you have not followed this tutorial, you can download the sMap file from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial7/tutorial5.blended.smap.bin.

- First of all, open a command-line interface in the folder containing the `tutorial5.blended.smap.bin` file.
- Open the file with Plot-sMap by typing **`Plot-sMap`** `-s tutorial5.blended.smap.bin` and pressing enter. The full description of the command line options for Plot-sMap is in 6.7.2; here, we are only using the `-s` option to specify the sMap file.
- The Plot-sMap user interface will open. You can navigate this (command-line) user interface using the arrow keys on your keyboard; you can generally activate a field for editing by pressing enter (a detailed description of every option is in 6.7.1).
- To have an idea of what we're working with, let's create first a plot with the default settings. To do so, move with the arrows to the `Plot…` button and press enter. A dialog will open requesting a name for the output file. If you want, you can change the default filename; otherwise just press enter to create the plot.
- If you used the default filename, a PDF file named `tutorial5.blended.smap.pdf` will have been created. Open this file with a PDF viewer.
- A first improvement to the plot would be making the tree wider (so that the shorter branches are still visible). To do this, highlight the `Plot width` field, press enter and replace the default value (500) with 750. Press enter again once you finish editing the field. Use again the `Plot` button to update the PDF plot, and check the results in your PDF viewer.
- The taxon labels in the tree represent species names, thus it would be appropriate to have them written in italics. Highlight the `Font family` field and press enter to reveal a list of available fonts (to which you can add your own custom font – see 6.7.1.6). Use the up and down arrows to select the `OpenSans-Italic.ttf` font and press enter to confirm (you can press the escape key to cancel). Again, update the plot and look at it in the PDF viewer. Note that the text in the legend for the state colours is now also in italics. It is not possible in Plot-sMap to change this font independently of the font used for the taxa labels (but you can use a vector editing program to do this).
- The default red and cyan colours make for a high contrast, but they are not exceptionally pretty. To change these, highlight the `State colours` field and press enter. A list of the states and the associated colours will open (note: due to the colour limitations of a console program, the colours displayed here will not necessarily be a good representation of the true colour that will be used in the plot – especially if unusual colour themes are used for the console). Make sure that the `A` state is highlighted, and press spacebar to edit the colour RGB values. For a light green colour, enter these RGB values: red 182, green 241, blue 200

(or choose your favourite colour). You can use the up and down arrows to move from one colour component to the next one. Press enter once you are satisfied with the chosen colour. Do the same for the `P` state colour. For a darker green, use red 34, green 177, blue 76. Press enter again once you finish editing the colours. Update the plot and you will see that the colours will change. If you are not satisfied with the look, you can change them again.

- Even though this tree is not time-calibrated, it can often be useful to plot a time scale along with the stochastic map. To do this, highlight the `Scale axis` field and press enter to enable it. If you also enable the `Scale grid` field, vertical lines at regular intervals will also be plotted.
- You can play with the various options (described in detail in 6.7.1) to see what effect they have on the plot.

This was the last tutorial involving the Cerataphidini dataset. In the next dataset we will use the Pontederiaceae dataset to look at two characters and whether they are best modelled as being dependent or independent of each other.

5.9 Analysing multiple characters (dependent vs independent)

In this tutorial we will analyse the Pontederiaceae dataset to determine whether the flower morphology and self-incompatibility are best modelled as dependent or independent characters.

You can download the files for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial8/Tutorial8.zip; extract them in a suitable folder.

- The `Pontederiaceae.tre` file contains the consensus (clocklike) tree.
- `Pontederiaceae.treedist` contains 1'000 tree samples from the posterior distribution.
- The `Pontederiaceae.txt` file contains the character state data for the two characters. The first column contains the data for the flower morphology (which can be **T**ristylous, **E**nantiostylous or **M**onomorphic). Note in particular that in the original dataset (Kohn *et al.*, 1996) *M. cyanea* is marked as uncertain between enantiostylous and monomorphic, thus the ambiguous state has been written as `{E:1,M:1}`. The second column contains the data for the self-incompatibility (**C**ompatible or **I**ncompatible).
- `Pontederiaceae_flower.txt` contains the states for the flower morphology character only.
- `Pontederiaceae_self.txt` contains the states for the self-incompatibility only.
- The `Pontederiaceae_self.model.ML.ER.nex`, `Pontederiaceae_self.model.Bayes.ER.nex` and `Pontederiaceae_self.model.Bayes.ARD.nex` files contain model specifications for the self-incompatibility character (see below).
- The `flower_models` folder contains model specifications for the flower morphology character (see below).
- The `Pontederiaceae_flower_ML_WIN.bat`, `Pontederiaceae_flower_ML_UNIX.sh`, `Pontederiaceae_flower_Bayes_WIN.bat` and `Pontederiaceae_flower_Bayes_UNIX.sh` files contain scripts to automate the analyses for the flower morphology character (see below).
- The `Pontederiaceae_dep.model.ML.ARD.nex`, `Pontederiaceae_dep.model.ML.ER.nex`, `Pontederiaceae_dep.model.ML.SYM.nex`, `Pontederiaceae_dep.model.Bayes.ARD.nex`, `Pontederiaceae_dep.model.Bayes.ER.nex`,

`Pontederiaceae_dep.model.Bayes.SYM.nex` files contain model specifications for the analysis considering the two characters as dependent on each other (see below).

We will consider multiple modelling hypotheses. At first, we will treat the two characters as independent (with ER, ARD, SYM, ORD/ER, ORD/ARD and ORD/SYM models for flower morphology and ER and ARD models for the self-incompatibility). Then, we will consider them as dependent on each other (with ER, ARD and SYM models). For each hypothesis, we will first run a maximum-likelihood (ML) analysis, and then use the ML estimates for the transition rates to inform the priors for a Bayesian analysis, also computing marginal likelihoods.

When considering the characters as independent, we will deal with them one at a time and, which will allow us to consider all combinations of models with fewer computations (since the likelihood of two independent characters is just the sum of the likelihoods for each character).

5.9.1 Independent characters: self-incompatibility

For the self-incompatibility characters, we will consider the ER and ARD models. For each, we will first run a maximum-likelihood analysis, to obtain ML estimates for the rates, and then we will use these estimates to inform the priors for a Bayesian analysis, in which we will compute the marginal likelihood.

To run the ARD analysis:

- Open a command line window and move to the folder where you have extracted the tutorial files. Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_self_ML_ARD`**.
- Run the ML sMap analysis by typing `sMap -t Pontederiaceae.tre -d Pontederiaceae_self.txt -o Plants_self_ML_ARD/tutorial8 -n 1000` and pressing enter. This command is similar to the ones used in previous tutorials.
- Take note of the ML estimates for the rates (which you can also find in the `tutorial8.mean.params.txt` file in the output folder), that should be around 2.822 for the $I \rightarrow C$ rate (which translates to a `LogNormal(1.03744585343062, 1)` prior) and 0.288 for the $C \rightarrow I$ rate (which corresponds to a `LogNormal(-1.24479479884619, 1)` prior). These priors are specified in the `Pontederiaceae_self.model.Bayes.ARD.nex` file.
- Create a new folder for the Bayesian analysis called **`Plants_self_Bayes_ARD`**.
- Run the Bayesian analysis estimating marginal likelihoods by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_self.txt -o Plants_self_Bayes_ARD/tutorial8 -n 1000 -i Pontederiaceae_self.model.Bayes.ARD.nex -ss` and pressing enter. This command is similar to the ones used in previous tutorials (don't forget to add the `--max-cov=1` switch if necessary, as explained in note 4 at page 29**Error! Bookmark not defined.**).
- Take note of the ln-marginal likelihood value, which should be around -9.85.

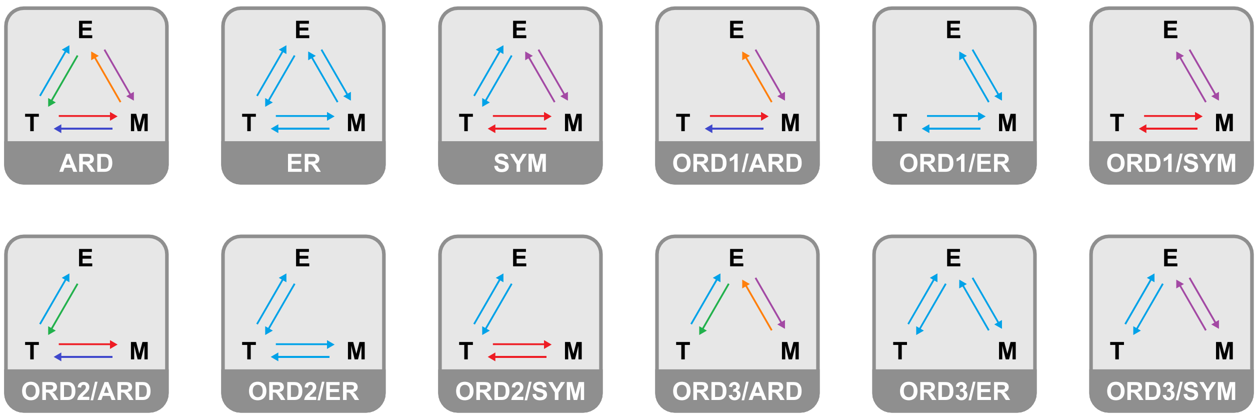
To run the ER analysis:

- Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_self_ML_ER`**.
- Run the ML sMap analysis by typing `sMap -t Pontederiaceae.tre -d Pontederiaceae_self.txt -o Plants_self_ML_ER/tutorial8 -n 1000 -i Pontederiaceae_self.model.ML.ER.nex` and pressing enter. This command is similar to the ones used in previous tutorials.

- Take note of the ML estimates for the rate, which should be around 0.248 (that translates to a `LogNormal(-1.39432653281715, 1)` prior). This prior is specified in the `Pontederiaceae_self.model.Bayes.ER.nex` file.
- Create a new folder for the Bayesian analysis called **`Plants_self_Bayes_ER`**.
- Run the Bayesian analysis estimating marginal likelihoods by typing **`sMap`** `-t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_self.txt -o` **`Plants_self_Bayes_ER`**`/tutorial8 -n 1000 -i Pontederiaceae_self.model.Bayes.ER.nex -ss` and pressing enter. This command is similar to the ones used in previous tutorials (don't forget to add the `--max-cov=1` switch if necessary, as explained in note 4 at page 29**Error! Bookmark not defined.**).
- Take note of the ln-marginal likelihood value, which should be around -10.79.

5.9.2 Independent characters: flower morphology

We will now run a similar analysis for the flower morphology character. However, since flower morphology is a 3-state character, we will consider more models (ER, ARD, SYM, and three sets of ordered models ORD1/ER, ORD1/ARD, ORD1/SYM, ORD2/ER, ORD2/ARD, ORD2/SYM, ORD3/ER, ORD3/ARD and ORD3/SYM). These models are presented in the following figure (the rates for transitions denoted by arrows of the same colour are considered to be equal):



These are by no means all possible models for a three-state character, but we will only use these for the sake of saving time.

To run the ARD analysis:

- Open a command line window and move to the folder where you have extracted the tutorial files. Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_flower_ML_ARD`**.
- Run the ML sMap analysis by typing **`sMap`** `-t Pontederiaceae.tre -d Pontederiaceae_flower.txt -o` **`Plants_flower_ML_ARD`**`/tutorial8 -n 1000` and pressing enter. This command is similar to the ones used in previous tutorials.
- Take note of the ML estimates for the rates, which should be around the ones shown in the following table. We can use these values to inform the priors for the Bayesian analysis; for the $T \rightarrow E$ and $E \rightarrow T$ rates, which have a ML estimate of 0, we will use an exponential prior with $\lambda = 100$, instead of the usual log-normal prior (see 6.1.1.2.1 for a list of all the available priors). These priors are specified in the `Pontederiaceae_flower.model.Bayes.ARD.nex` file which is located in the `flower_models` folder.

Rate	ML estimate	Prior

$T \rightarrow M$	2.737	`LogNormal(1.00686243005854, 1)`
$T \rightarrow E$	0	`Exponential(100)`
$M \rightarrow T$	2.164	`LogNormal(0.771958360984235, 1)`
$M \rightarrow E$	0.937	`LogNormal(-0.0650719967437148, 1)`
$E \rightarrow T$	0	`Exponential(100)`
$E \rightarrow M$	1.01	`LogNormal(0.00995033085316809, 1)`

- Create a new folder for the Bayesian analysis called **`Plants_flower_Bayes_ARD`**.
- Run the Bayesian analysis estimating marginal likelihoods by typing **`sMap`** `-t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_flower.txt -o` **`Plants_flower_Bayes_ARD`**`/tutorial8 -n 1000 -i flower_models/Pontederiaceae_flower.model.Bayes.ARD.nex -ss` and pressing enter. This command is similar to the ones used in previous tutorials (don't forget to add the `--max-cov=1` switch if necessary, as explained in note 4 at page 29).
- Take note of the ln-marginal likelihood value, which should be around -21.69.

For the remaining 11 models, we will use some scripting to save time, instead of typing every command. Depending on your operating system, you will want to use the script files ending with `WIN.bat` (on Windows) or with `UNIX.sh` (on macOS and Linux).

To run the maximum-likelihood analyses:

- On Windows:
  - o Type in the command line `Pontederiaceae_flower_ML_WIN.bat` and press enter.
- On Linux/macOS:
  - o Type in the command line `./Pontederiaceae_flower_ML_UNIX.sh` and press enter.

The contents of the scripts are very similar and will take care of creating the output folder for each analysis and running them. Wait until the 11 ML analyses finish.

Now, you should inspect the `tutorial8.mean.params.txt` files located in each of the 11 output folders and obtain the ML estimates for the rates from there; the logarithm of these estimates would then be used to determine the parameter for the log-normal priors for the Bayesian analyses. If you want to save time, the priors for these analyses are available in the Bayesian model files in the `flower_models` folder (where the ML estimate was smaller than 0.01, an Exponential(100) prior was used).

To run the Bayesian analyses we will again use a script:

- On Windows:
  - o Type in the command line `Pontederiaceae_flower_Bayes_WIN.bat` and press enter.
- On Linux/macOS:
  - o Type in the command line `./Pontederiaceae_flower_Bayes_UNIX.sh` and press enter.

Again, the script will create folders for the output files and perform the Bayesian analyses, also computing marginal likelihoods. Wait until everything finishes (this may take a while).

After all is done, you can find the computed ln-marginal likelihoods in the `tutorial8.marginal.likelihood.txt` files in the output folders. They should be similar to these values:

Model	ln-marginal likelihood
ER	-22.63
SYM	-20.79
ORD1/ARD	-21.77
ORD1/ER	-20.97
ORD1/SYM	-20.78
ORD2/ARD	-23.49
ORD2/ER	-23.65
ORD2/SYM	-22.13
ORD3/ARD	-24.97
ORD3/ER	-24.72
ORD3/SYM	-23.31

We can now use these values together with the ones we computed for the self-incompatibility character to compute the marginal likelihoods for all combinations of character models:

Flower morphology model	Self-incompatibility model	Flower ln-marginal likelihood	Self-incompatibility ln-marginal likelihood	Overall ln-marginal likelihood
ARD	ARD	-21.69	-9.85	-31.54
ER	ARD	-22.63	-9.85	-32.48
SYM	ARD	-20.79	-9.85	-30.64
ORD1/ARD	ARD	-21.77	-9.85	-31.62
ORD1/ER	ARD	-20.97	-9.85	-30.82
ORD1/SYM	ARD	-20.78	-9.85	-30.63
ORD2/ARD	ARD	-23.49	-9.85	-33.34
ORD2/ER	ARD	-23.65	-9.85	-33.5
ORD2/SYM	ARD	-22.13	-9.85	-31.98
ORD3/ARD	ARD	-24.97	-9.85	-34.82
ORD3/ER	ARD	-24.72	-9.85	-34.57
ORD3/SYM	ARD	-23.31	-9.85	-33.16
ARD	ER	-21.69	-10.79	-32.48
ER	ER	-22.63	-10.79	-33.42
SYM	ER	-20.79	-10.79	-31.58
ORD1/ARD	ER	-21.77	-10.79	-32.56
ORD1/ER	ER	-20.97	-10.79	-31.76
ORD1/SYM	ER	-20.78	-10.79	-31.57
ORD2/ARD	ER	-23.49	-10.79	-34.28
ORD2/ER	ER	-23.65	-10.79	-34.44
ORD2/SYM	ER	-22.13	-10.79	-32.92
ORD3/ARD	ER	-24.97	-10.79	-35.76
ORD3/ER	ER	-24.72	-10.79	-35.51
ORD3/SYM	ER	-23.31	-10.79	-34.1

5.9.3 Dependent characters

We will now consider the two characters as dependent on each other, which will allow sMap to merge the 2-state and 3-state characters into a single 6-state character. We will consider three models in this case: ARD, ER and SYM. As usual, we will run a maximum-likelihood analysis first, followed by a Bayesian analysis computing marginal likelihoods.

To perform the analyses for the ARD model:

- Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_dep_ML_ARD`**.
- Run the ML sMap analysis by typing **`sMap`** `-t Pontederiaceae.tre -d Pontederiaceae.txt -o` **`Plants_dep_ML_ARD`**`/tutorial8 -n 1000 -i Pontederiaceae_dep.model.ML.ARD.nex` and pressing enter. This command is similar to the ones used in previous tutorials.
- Take note of the ML estimates for the rates, which should be around the ones shown in the following table. As before, we can use these values to inform the priors for the Bayesian analysis; for the rates that have a ML estimate of 0, we will use an exponential prior with $\lambda = 100$. These priors are specified in the `Pontederiaceae_dep.model.Bayes.ARD.nex` file.

Rate	ML estimate	Prior
$(T, I) \to (M, C)$	2.691	`LogNormal(0.989912871744769, 1)`
$(T, C) \to (M, C)$	2.411	`LogNormal(0.880041599199034, 1)`
$(M, I) \to (E, I)$	9.803	`LogNormal(2.28268846127959, 1)`
$(M, C) \to (T, I)$	1.026	`LogNormal(0.0256677467485778, 1)`
$(M, C) \to (T, C)$	0.934	`LogNormal(-0.0682788407532944, 1)`
$(M, C) \to (E, C)$	0.882	`LogNormal(-0.125563222975346, 1)`
$(E, I) \to (T, C)$	3.168	`LogNormal(1.15310047395218, 1)`
$(E, I) \to (M, C)$	2.239	`LogNormal(0.806029337616618, 1)`
$(E, I) \to (E, C)$	3.015	`LogNormal(1.10359983017915, 1)`
$(E, C) \to (M, C)$	0.918	`LogNormal(-0.0855578883616465, 1)`
Others	0	`Exponential(100)`

- Create a new folder to hold the results of the Bayesian analysis, naming it **`Plants_dep_Bayes_ARD`**.
- Run the Bayesian sMap analysis by typing **`sMap`** `-t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae.txt -o` **`Plants_dep_Bayes_ARD`**`/tutorial8 -n 1000 -i Pontederiaceae_dep.model.Bayes.ARD.nex -ss --max-cov=1 --min-ess=1 --min-samples=10000 --ss-estimate-steps` and pressing enter. This command is similar to the ones used in previous tutorials, but has a few additions to speed up the convergence of the chains (sometimes, at the expense of accuracy):
  - `--max-cov=1`: as explained in note 4 at page 29, this disables the coefficient-of-variation convergence criteria
  - `--min-ess=1`: this disables the ESS convergence criterion.

    - `--min-samples=10000`: this increases the minimum number of samples required for each chain from the default 2'000 (twice the number of stochastic mapping histories) to 10'000. We use this since the number of samples is the only convergence criterion we are leaving active, in order to obtain a decent number of samples.
    - `--ss-estimate-steps`: this enables proposal step-size estimation for each step in the stepping-stone analysis. In practice, this increases the duration of the burn-in phase of each stepping-stone step, but allows the chain to converge faster.
- We are using these variations because with 35 parameters to sample, for the analysis to converge it would take an amount of time that would be unreasonable for a tutorial. In a real analysis (when you do not have such time constraints), you would only leave the `--ss-estimate-steps` enabled (as this increases the convergence rate without affecting accuracy) and skip the other switches.
- Once the analysis finishes, take note of the ln-marginal likelihood, which should be around -24.85.

To perform the analyses for the ER model:

- Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_dep_ML_ER`**.
- Run the ML sMap analysis by typing `sMap -t Pontederiaceae.tre -d Pontederiaceae.txt -o Plants_dep_ML_ER/tutorial8 -n 1000 -i Pontederiaceae_dep.model.ML.ER.nex` and pressing enter. This command is similar to the ones used in previous tutorials.
- Take note of the ML estimate for the rates, which should be around 0.252, which translates to a `LogNormal(-1.37832619147071, 1)` prior. This prior is specified in the `Pontederiaceae_dep.model.Bayes.ER.nex` file.
- Create a new folder to hold the results of the Bayesian analysis, naming it **`Plants_dep_Bayes_ER`**.
- Run the Bayesian sMap analysis by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae.txt -o Plants_dep_Bayes_ER/tutorial8 -n 1000 -i Pontederiaceae_dep.model.Bayes.ER.nex -ss` and pressing enter. This command is similar to the ones used in previous tutorials (don't forget to add the `--max-cov=1` switch if necessary, as explained in note 4 at page 29). Since we are only sampling 6 parameters, it should be possible for the chains to converge in a short time even without the "tricks" that we used for the ARD model.
- Once the analysis finishes, take note of the ln-marginal likelihood, which should be around -32.36.

Finally, to perform the analyses for the SYM model:

- Create a new folder to hold the results of the maximum-likelihood analysis, naming it **`Plants_dep_ML_SYM`**.
- Run the ML sMap analysis by typing `sMap -t Pontederiaceae.tre -d Pontederiaceae.txt -o Plants_dep_ML_SYM/tutorial8 -n 1000 -i Pontederiaceae_dep.model.ML.SYM.nex` and pressing enter. This command is similar to the ones used in previous tutorials.
- Take note of the ML estimates for the rates, which should be around the ones shown in the following table. As before, we wil use these values to inform the priors for the Bayesian analysis; for the rates that have a ML estimate of 0, we will use an exponential prior with $\lambda = 100$. These priors are specified in the `Pontederiaceae_dep.model.Bayes.SYM.nex` file.

Rate	ML estimate	Prior
$(T,I) \rightarrow (M,C)$ $(M,C) \rightarrow (T,I)$	1.305	`LogNormal(0.266203040774657, 1)`
$(T,C) \rightarrow (M,C)$ $(M,C) \rightarrow (T,C)$	1.121	`LogNormal(0.114221144090023, 1)`
$(M,I) \rightarrow (E,I)$ $(E,I) \rightarrow (M,I)$	9.200	`LogNormal(2.21920348405499, 1)`
$(M,C) \rightarrow (E,C)$ $(E,C) \rightarrow (M,C)$	1.056	`LogNormal(0.0544881852840698, 1)`
Others	0	`Exponential(100)`

- Create a new folder to hold the results of the Bayesian analysis, naming it **`Plants_dep_Bayes_SYM`**.
- Run the Bayesian sMap analysis by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae.txt -o Plants_dep_Bayes_SYM/tutorial8 -n 1000 -i Pontederiaceae_dep.model.Bayes.SYM.nex -ss --max-cov=1 --min-ess=1 --min-samples=10000 --ss-estimate-steps` and pressing enter. Here we need to sample 20 parameters, therefore we are using the same arguments that we used to speed up convergence in the ARD analysis.
- Once the analysis finishes, take note of the ln-marginal likelihood, which should be around -25.18.

We can now compare all the models that we have considered (including the ones where the characters were independent) and compute posterior probabilities for each model (assuming a uniform prior for the models, see 1.2.3.3):

Character relationship	Flower morphology model	Self-incompatibility model	ln-marginal likelihood	Posterior probability
Independent	ARD	ARD	-31.54	0.07%
	ER	ARD	-32.48	0.03%
	SYM	ARD	-30.64	0.18%
	ORD1/ARD	ARD	-31.62	0.07%
	ORD1/ER	ARD	-30.82	0.15%
	ORD1/SYM	ARD	-30.63	0.18%
	ORD2/ARD	ARD	-33.34	0.01%
	ORD2/ER	ARD	-33.5	0.01%
	ORD2/SYM	ARD	-31.98	0.05%
	ORD3/ARD	ARD	-34.82	0.00%
	ORD3/ER	ARD	-34.57	0.00%
	ORD3/SYM	ARD	-33.16	0.01%
	ARD	ER	-32.48	0.03%
	ER	ER	-33.42	0.01%
	SYM	ER	-31.58	0.07%

	ORD1/ARD	ER	-32.56	0.03%
	ORD1/ER	ER	-31.76	0.06%
	ORD1/SYM	ER	-31.57	0.07%
	ORD2/ARD	ER	-34.28	0.00%
	ORD2/ER	ER	-34.44	0.00%
	ORD2/SYM	ER	-32.92	0.02%
	ORD3/ARD	ER	-35.76	0.00%
	ORD3/ER	ER	-35.51	0.00%
	ORD3/SYM	ER	-34.1	0.01%
	ARD		-24.85	57.55%
Dependent	ER		-32.36	0.03%
	SYM		-25.18	41.37%

All models except the ARD and SYM models for dependent characters can be essentially ignored, as they have negligible posterior probabilities. We can now blend the ARD and SYM analyses using the Blend-sMap utility:

- First, create a new folder named `Plants_blended` to host the results of the analysis.
- To blend the analyses, type: `Blend-sMap -o Plants_blended/tutorial8.blended.smap.bin -n 2000 -s Plants_dep_Bayes_ARD/tutorial8.smap.bin,57.55 -s Plants_dep_Bayes_SYM/tutorial8.smap.bin,41.37` and press enter. This command is similar to the ones used previously.

We will create plots of the blended analysis with the Plot-sMap utility.

- To create a plot for both characters at once, type: `Plot-sMap -s Plants_blended/tutorial8.blended.smap.bin -batch` and press enter. This will create a file called `tutorial8.blended.smap.pdf` in the same folder as the sMap file.
- To create a plot for the flower morphology, type: `Plot-sMap -s Plants_blended/tutorial8.blended.smap.bin -batch -c 0 -o Plants_blended/tutorial8.blended.smap.flower.pdf` and press enter. The main differences between this command and the previous one are:
    - `-c 0`: this argument enables plotting only of the first character (in this case, the flower morphology)
    - `-o Plants_blended/tutorial8.blended.smap.flower.pdf`: this specifies the name of the output file
- To create a plot for the self-incompatibility, type: `Plot-sMap -s Plants_blended/tutorial8.blended.smap.bin -batch -c 1 -o Plants_blended/tutorial8.blended.smap.self.pdf` and press enter.

You can now look at the plots in the output folder to see the results of the analysis.

This is the end of this tutorial. In the next tutorial, we will model the self-incompatibility character as being conditioned on the flower morphology character.

5.10 Analysing multiple characters (conditioned)

In this tutorial we will analyse again the Pontederiaceae dataset considering the self-incompatibility character as conditioned on the flower morphology.

You can download the files for this tutorial from https://github.com/arklumpus/sMap/raw/master/Tutorials/Tutorial9/Tutorial9.zip; extract the files and the `cond_models` folder in the same folder where you have the files for the previous tutorial.

As before, we will consider 12 models for the flower morphology character (see 5.9.2). We will use the ML estimates for the rates from the previous tutorial to inform the priors for the analyses in this tutorial. The model files specifying these priors are located in the folder you downloaded. If you open one of these files, you will see that the `Dependency` section at the start specifies that character 1 (self-incompatibility) should be dependent on character 0 (flower morphology), with a flat Dirichlet prior for the conditioned probabilities.

As before, we will use a script to run the Bayesian analyses:

- On Windows:
    - Type in the command line `Pontederiaceae_cond_Bayes_WIN.bat` and press enter.
- On Linux/macOS:
    - Type in the command line `./Pontederiaceae_cond_Bayes_UNIX.sh` and press enter.

As before, the script will create the output folders and run the analyses. It also specifies the arguments to speed up convergence that we used before (see 5.9.3). Wait for the 12 Bayesian analyses to finish (this will probably take quite a long time, unfortunately).

The marginal likelihoods will be reported in the `tutorial9.marginal.likelihood.txt` files in each output folder, and should be similar to the ones in the following table; we can use them to compute posterior probabilities for the models:

Model	ln-marginal likelihood	Posterior probability
ARD	-31.64	10.06%
ER	-32.67	3.59%
SYM	-30.81	23.07%
ORD1/ARD	-31.73	9.20%
ORD1/ER	-31.01	18.89%
ORD1/SYM	-30.80	23.31%
ORD2/ARD	-33.40	1.73%
ORD2/ER	-33.76	1.21%
ORD2/SYM	-32.14	6.10%
ORD3/ARD	-34.74	0.45%
ORD3/ER	-34.79	0.43%
ORD3/SYM	-33.28	1.95%

We can now blend the analyses using Blend-sMap:

- First, create a new folder named **`Plants_cond_blended`** to host the results of the analysis.
- To blend the analyses, type: **`Blend-sMap`** `-o` **`Plants_cond_blended`**`/tutorial9.blended.smap.bin -n 10000 -s` **`Plants_cond_Bayes_ARD`**`/tutorial9.smap.bin,10.06 -s` **`Plants_cond_Bayes_ER`**`/tutorial9.smap.bin,3.59 -s`

Plants_cond_Bayes_SYM/tutorial9.smap.bin,23.07 -s **Plants_cond_Bayes_ORD1_ARD**/tutorial9.smap.bin,9.20 -s **Plants_cond_Bayes_ORD1_ER**/tutorial9.smap.bin,18.89 -s **Plants_cond_Bayes_ORD1_SYM**/tutorial9.smap.bin,23.31 -s **Plants_cond_Bayes_ORD2_ARD**/tutorial9.smap.bin,1.73 -s **Plants_cond_Bayes_ORD2_ER**/tutorial9.smap.bin,1.21 -s **Plants_cond_Bayes_ORD2_SYM**/tutorial9.smap.bin,6.10 -s **Plants_cond_Bayes_ORD3_ARD**/tutorial9.smap.bin,0.45 -s **Plants_cond_Bayes_ORD3_ER**/tutorial9.smap.bin,0.43 -s **Plants_cond_Bayes_ORD3_SYM**/tutorial9.smap.bin,1.95 and press enter. This command is similar to the ones used previously. In this case we have specified all the models, though, to save time, it would also have been possible to ignore the ones with the lowest probabilities (e.g. $< 5\%$).

Finally, we will create plots of the blended analysis with the Plot-sMap utility.

- To create a plot for both characters at once, type: **Plot-sMap** -s **Plants_cond_blended**/tutorial9.blended.smap.bin -batch and press enter. This will create a file called `tutorial9.blended.smap.pdf` in the same folder as the sMap file.
- To create a plot for the flower morphology, type: **Plot-sMap** -s **Plants_cond_blended**/tutorial9.blended.smap.bin -batch -c 0 -o **Plants_cond_blended**/tutorial9.blended.smap.flower.pdf and press enter.
- To create a plot for the self-incompatibility, type: **Plot-sMap** -s **Plants_cond_blended**/tutorial9.blended.smap.bin -batch -c 1 -o **Plants_cond_blended**/tutorial9.blended.smap.self.pdf and press enter.

You can now look at the plots in the output folder to see the results of the analysis.

This is the end of this tutorial. Note that it would also be possible to include the models we have analysed here in the model comparison of the previous tutorial, thereby choosing whether the characters are best modelled as independent, dependent, or conditioned.

5.11 Testing for correlation between characters

In this tutorial we will perform a D-test (Huelsenbeck, Nielsen and Bollback, 2003) on the Pontederiaceae dataset to determine whether the self-incompatibility and flower morphology evolve in a correlated fashion.

The D-test is a "posterior-predictive check", which essentially tells whether the two characters show more correlation than would be expected under the model that has been used. This is different than the model fit analyses that we have performed in the previous tutorials, as the D-test does not provide any information about *how* this correlation could be modelled.

It is also important to highlight out that the D-test tests for *more correlation than the current model*. For example, with sMap it is possible to define models in which two characters are perfectly correlated (e.g. by modelling one as conditioned on the other, with conditioned probabilities all equal to 0 or 1); if a D-test is performed between these two characters, the results will not be significant (the posterior-predictive P will be around 0.5). This does not mean that the characters are not correlated, but rather that they are as correlated as the model predicts (i.e., perfectly correlated).

In this tutorial, we will use the same files as in tutorial 8 (see **5.9**), but for simplicity we will only consider the ER and ARD model for the flower morphology character (and we will re-use the priors and marginal likelihood estimates from that tutorial).

To run a D-test in sMap, it is necessary to set it up during the initial stochastic mapping analysis. In this tutorial we will thus start by repeating the sMap analyses for self-incompatibility and flower morphology; we will then blend the analyses for the different models, merge the analyses for the two characters and finally perform the D-test.

5.11.1 Self-incompatibility

For this character, we will run a Bayesian ARD and ER analysis, with the posterior-predictive sampling option enabled, and we will then blend these analyses according to model posterior probabilities that can be computed using the marginal likelihoods sampled in tutorial 8 (see **5.9**).

To run the ARD analysis:

- Create a new folder for the Bayesian analysis called **`Plants_self_PP_ARD`**.
- Run the Bayesian analysis sampling the posterior predictive distribution by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_self.txt -o Plants_self_PP_ARD/tutorial10 -n 1000 -i Pontederiaceae_self.model.Bayes.ARD.nex --pp 100` and pressing enter. This command is similar to the ones used in previous tutorials; the main difference is the -pp 100 option, which will cause 100 posterior predictive samples to be drawn for each of the 1000 parameter values that have been sampled. Each of these 100’000 sampled histories will be saved in the sMap output file (see below for more information on the impact of this on disk usage).
- Wait until the analysis finished. Remember that the log-marginal likelihood value for this model was around -9.85.

To run the ER analysis:

- Create a new folder for the Bayesian analysis called **`Plants_self_PP_ER`**.
- Run the Bayesian analysis sampling the posterior predictive distribution by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_self.txt -o Plants_self_PP_ER/tutorial10 -n 1000 -i Pontederiaceae_self.model.Bayes.ER.nex --pp 100` and pressing enter. This command is similar to the one used in the previous step.
- Wait until the analysis finished. Remember that the log-marginal likelihood value for this model was around -10.79.

To blend the two analyses:

- Compute the model posterior probabilities, which should be around 71.9% for the ARD model and 28.1% for the ER model.
- Blend the analyses by typing `Blend-sMap -o tutorial10_self.blended.smap.bin -n 2000 -s Plants_self_PP_ARD/tutorial10.smap.bin,71.9 -s Plants_self_PP_ER/tutorial10.smap.bin,28.1` and pressing enter. This command is similar to the ones used in previous tutorials.

5.11.2 Flower morphology

For this character too, we will run a Bayesian ARD and ER analysis, with the posterior-predictive sampling option enabled, and we will then blend these analyses according to model posterior probabilities that can be computed using the marginal likelihoods sampled in tutorial 8 (see **5.9**).

To run the ARD analysis:

- Create a new folder for the Bayesian analysis called **`Plants_flower_PP_ARD`**.
- Run the Bayesian analysis sampling the posterior predictive distribution by typing `sMap -t Pontederiaceae.treedist -T Pontederiaceae.tre -d`

`Pontederiaceae_flower.txt -o` **`Plants_flower_PP_ARD`**`/tutorial10 -n 1000 -i flower_models/Pontederiaceae_flower.model.Bayes.ARD.nex --pp 100` and pressing enter. This command is similar to the ones used previously.
- Wait until the analysis finished. Remember that the log-marginal likelihood value for this model was around -21.69.

To run the ER analysis:

- Create a new folder for the Bayesian analysis called **`Plants_flower_PP_ER`**.
- Run the Bayesian analysis sampling the posterior predictive distribution by typing **`sMap`** `-t Pontederiaceae.treedist -T Pontederiaceae.tre -d Pontederiaceae_flower.txt -o` **`Plants_flower_PP_ER`**`/tutorial10 -n 1000 -i flower_models/Pontederiaceae_flower.model.Bayes.ER.nex --pp 100` and pressing enter. This command is similar to the ones used in the previous steps.
- Wait until the analysis finished. Remember that the log-marginal likelihood value for this model was around -22.63.

To blend the two analyses:

- Compute the model posterior probabilities, which should be around 71.9% for the ARD model and 28.1% for the ER model.
- Blend the analyses by typing **`Blend-sMap`** `-o tutorial10_flower.blended.smap.bin -n 2000 -s` **`Plants_flower_PP_ARD`**`/tutorial10.smap.bin,71.9 -s` **`Plants_flower_PP_ER`**`/tutorial10.smap.bin,28.1` and pressing enter. This command is similar to the ones used in previous tutorials.

We now need to merge the analyses for the two characters. We can do so by using the Merge-sMap utility.

To merge the two blended analyses, type **`Merge-sMap`** `-o tutorial10.merged.smap.bin -n 2000 -s tutorial10_self.blended.smap.bin;0 -s tutorial10_flower.blended.smap.bin;0` and press enter. The arguments passed to Merge-sMap in this command are:

- `-o tutorial10.merged.smap.bin`: this argument defines the sMap output file.
- `-n 2000`: this argument defines the number of simulations that will be included in the output file (these will be drawn at random from the simulations in the input files).
- `-s tutorial10_self.blended.smap.bin;0`: this argument provides an input file and identifies which characters from that file should be included (in this case, character 0, which is the only character).
- `-s tutorial10_flower.blended.smap.bin;0`: same as above.

Finally, we will now perform the D-test itself using the Stat-sMap utility. To perform the D-test, type **`Stat-sMap`** `-s tutorial10.merged.smap.bin -t 0 1 tutorial10.dtest` and press enter. The arguments passed to Stat-sMap in this command are:

- `-s tutorial10.merged.smap.bin`: the input sMap file, which must contain simulations for more than one character.
- `-t 0 1 tutorial10.dtest`: this argument specifies that we want to perform a D-test between characters 0 and 1, and that we want to save the output of this test as a file called `tutorial10.dtest.pdf`.

Once the D-test is performed, Stat-sMap will print some statistics on the screen: the value of the D and d_{ij} statistics (see Huelsenbeck, Nielsen and Bollback, 2003) and the posterior-predictive P and

P_{ij} values. The lower these P and P_{ij} values are, the more significant is the correlation between the two characters, compared to what the model (which, in this case, assumes that they were independent) would predict. The P and D refer to the two characters as a whole, while d_{ij} and P_{ij} refer to specific pairs of states. The sign of the d_{ij} indicates the kind of correlation (if it is positive, the state pair is found more than was expected, while if it is negative, the state pair is rarer than expected).

A PDF file containing these statistics together with a plot of the distribution of the D statistic in the posterior-predictive samples is also produced.

In this case, it seems that there is some correlation between these two characters, but this is not overly significant.

A note about the posterior-predictive distribution sampling: this can be done in sMap either using the `--pp/--posterior-predictive` argument, as we have done here, or using the `--dt/--d-test` argument. The difference is the following:

- If the `--dt/--d-test` argument is used, the specified number of posterior-predictive histories is simulated, and for each history a D-test is performed, between all possible pairs of characters in the history. The results of these D-tests are saved, while the histories are discarded. This means that this argument is useful only when all the characters between which you want to perform a D-test are included in the same model. If, at a later time, you want to perform a D-test between one of these characters and a different character, you will have to run the whole analysis from scratch. These D-tests are preserved when blending histories for the same data using Blend-sMap, but are discarded when histories for different characters are merged with Merge-sMap.
- If the `--pp/--posterior-predictive` argument is specified, the posterior-predictive histories are sampled and saved in the sMap output file. This makes it possible to compute D-tests between completely separate sMap analyses (by merging them with Merge-sMap), but causes the sMap output file to be much bigger than with the other option. It also causes many input/output operations to be performed on the hard disk.

In summary, the `--pp/--posterior-predictive` argument is more versatile, but causes bigger output files, while the `--dt/--d-test` argument is more limited, but does not increase file size by much, compared to a standard sMap analysis.

6 Program description

In this section the various programs included in the sMap suite will be described, with an overview of the algorithms involved, command-line parameters and output files.

Command line parameters can be of two kinds: parameters that supply a value (highlighted in **blue**), which are used for example to specify input files, or switches (highlighted in **green**), which are used to turn specific features on or off.

For parameters that require a value, this value can be required (denoted as **`--parameter <value>`**) or optional (denoted as **`--parameter [value]`**).

When the path to a file (for input or output) is needed, it can be specified as a relative or absolute path; furthermore, it can be specified on any platform using both Windows and UNIX conventions (i.e. both / and \ can be used as directory separators – but note that on UNIX platforms \ may be interpreted as an escape character by the command line).

A switch can be enabled by specifying **`--switch`** or **`--switch+`** in the command line, and it can be disabled by **`--switch-`** (this is useful in the case of switches that are on by default).

6.1 sMap

The sMap program is the main program, which performs the actual stochastic mapping analyses.

The general workings of a stochastic mapping analysis have been described in 1.1; here we will focus on how they are implemented in sMap.

6.1.1 Parameter sampling

sMap implements two main approaches to parameter sampling: maximum-likelihood and Bayesian sampling.

6.1.1.1 Maximum-likelihood

In a maximum-likelihood (ML) analysis, the values of the parameter that maximise the likelihood function are determined, therefore a single set of parameters is used for all the simulations in the analysis. If multiple trees are provided, a different tree will be used for each simulation, but the parameters will only be optimised once (using the "mean tree" that must also be provided).

There are multiple strategies that can be used in sMap to obtain maximum-likelihood estimates for the parameters. To describe them, we will consider the likelihood function of **Figure 3** and see how the various strategies sample this function to arrive to the ML estimate.

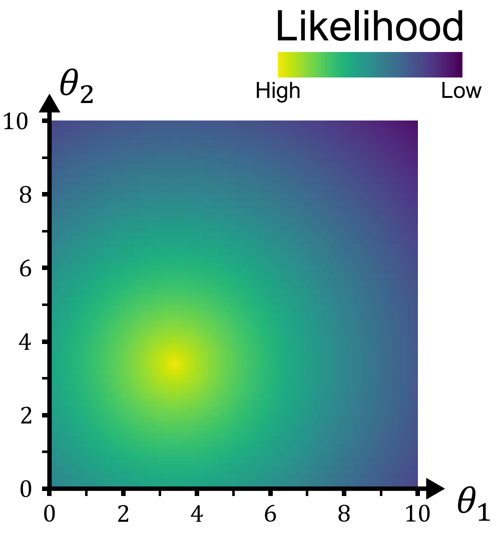


Figure 3. A likelihood function that depends on two parameters, $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$. The colour of the area represents the value of the likelihood: purple-blue is a low value, while yellow-green is a high value. The true maximum value of the likelihood is at $\boldsymbol{\theta_1 = \theta_2 = 3.39}$.

The strategies to employ can be specified to sMap by using the `-m` command line parameter (see 6.1.5). Each strategy has several parameters, which specify how it behaves.

In particular, one of the parameters for each strategy determines whether a plot showing the progress of the estimation should be printed in the command line or not. Values of `true` or `plot` cause the plot to be shown, values of `false` or `noplot` indicate that no plot should be produced. Showing a plot can help because it gives an idea of the state of the analysis, but on the other hand it also causes the analysis to be slightly slower.

6.1.1.1.1 Sampling

In the "sampling" approach, equally spaced points in the parameter space are sampled. This strategy has the following syntax: **`Sampling(min,max,resolution,plot)`**; `min` determines the lower bound of the range to test (must be greater than 0), `max` determines the upper bound (must be greater than `min`), `resolution` determines the spacing between the samples and `plot` determines whether a plot showing the progress of the analysis should be printed in the command line or not.

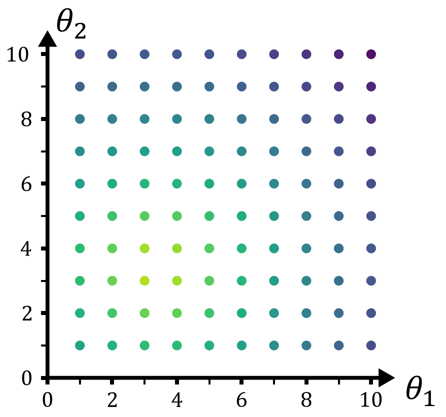


Figure 4. The values of parameters $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$ that are sampled by a `Sampling(1, 10, 1, plot)` strategy. The maximum-likelihood estimate would be $\boldsymbol{\theta_1 = \theta_2 = 3}$.

For example, a `Sampling(1,10,1,plot)` strategy specifies that, for each parameter, every point value should be sampled starting from 1 and incrementing by 1, up to 10 (i.e. 1, 2, 3 …, 8, 9, 10). In the case of multiple parameters (such as the likelihood function of **Figure 3**), all possible parameter value combinations are sampled (in this case, the 100 couples {1, 1}, {1, 2}, {1, 3}, …, {2, 1}, …, {10, 1}, …, {10, 10}). This means that the number of likelihoods to compute grows exponentially with the number of parameters, thus this approach is only feasible for a small number of parameters. **Figure 4** shows the values that would be sampled by this strategy when applied to the likelihood function of **Figure 3**; the maximum-likelihood estimate obtained with this approach is $\boldsymbol{\theta_1 = \theta_2 = 3}$ (because, with a resolution of 1, this is the closest point to the peak).

6.1.1.1.2 Iterative sampling

The "iterative sampling" approach is similar to the sampling approach, but for each iteration the parameters are fixed to the current ML estimate and optimised one at a time. In the next iteration, each parameter is again optimised, until a stationary state is reached where it is no longer possible to increase the likelihood by changing the value of any parameter. The number of likelihood computations for each iteration in this approach grows linearly with the number of parameters, which makes it useful for relatively large sets of parameters (though the number of iterations that are necessary may increase as well); the parameter space is however not completely sampled, therefore the algorithm may end up on a local maximum.

The syntax for this strategy is similar to the previous one: **`IterativeSampling(min,max,resolution,threshold,plot)`**, where the `min`, `max`, `resolution` and plot parameters have the same meaning. The `threshold` parameter provides a stopping criterion: if an iteration finishes without improving the likelihood by at least `threshold`, it is assumed that a stationary point has been reached.

Figure 5 shows how an `IterativeSampling(1,10,1,0.001,plot)` strategy would behave on the likelihood function of **Figure 3**.

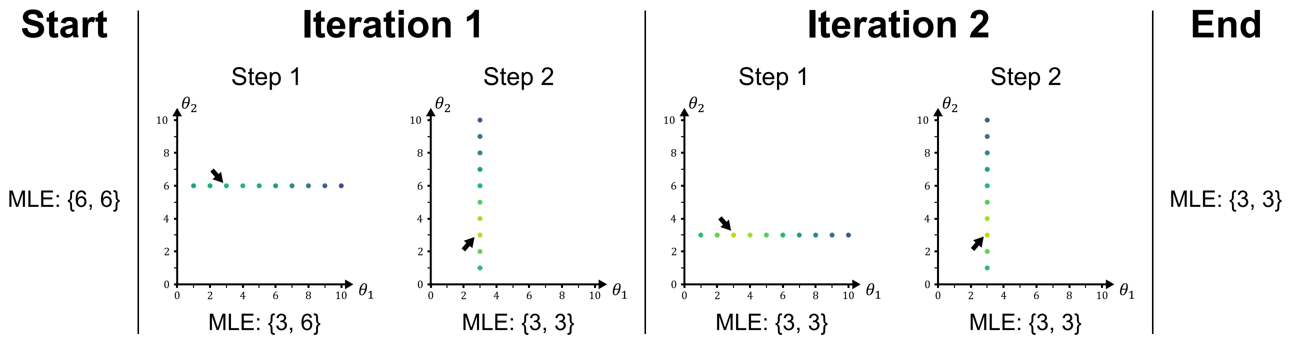


Figure 5. The process of estimating maximum likelihood values for the likelihood function in ***Figure 3*** using the iterative sampling strategy. The process is initialised with both parameters set at the midpoint of the estimation range ({6, 6}). In the first step of the first iteration, $\boldsymbol{\theta_2}$ is fixed at its current estimate of 6, and the likelihood is computed for the different values of $\boldsymbol{\theta_1}$. The value with the highest likelihood (3, arrow) is chosen as the new value for $\boldsymbol{\theta_1}$; thus, at the end of this step the maximum-likelihood estimate (MLE) is $\boldsymbol{\theta_1 = 3}$, $\boldsymbol{\theta_2 = 6}$. In the second step, $\boldsymbol{\theta_1}$ is kept fixed at 3, and the likelihood is computed for the different values of $\boldsymbol{\theta_2}$. The value with the highest likelihood (3, arrow) is chosen as the new value for $\boldsymbol{\theta_2}$; thus, at the end of this step (and of the first iteration) the MLE is $\boldsymbol{\theta_1 = 3}$, $\boldsymbol{\theta_2 = 3}$. The process is repeated again for the second iteration; since the MLE at the end of the second iteration has not changed, we have reached a stationary point and the process ends.

The results are the same as with the sampling approach, but only 40 likelihoods have been computed, instead of 100. Furthermore, the iterative sampler is optimised to run on multiple threads (the number of which can be specified with the `--num-threads`/`--nt` parameter of sMap (see 6.1.5).

6.1.1.1.3 Random walk

Under the "random walk" approach, starting from an arbitrary set of parameter values, random updates to the parameters are proposed. If the likelihood under the new values is greater than the previous likelihood, the update is always accepted; otherwise, it is accepted with a probability that is proportional to the ratio of the logarithm of the likelihoods. The set of parameters with the maximum likelihood that has been visited is recorded and constitutes the output of the algorithm. The algorithm can be repeated multiple times, until no change in the maximum-likelihood value is observed. The number of likelihood computations required for this approach does not depend on the number of parameters, therefore it can be useful to estimate an arbitrarily large number of parameters. The performance (in terms of the maximum likelihood estimate) is however usually worse than either sampling approach.

The syntax for this strategy is **`RandomWalk(steps,criterion,threshold,plot)`**; `steps` determines how many random proposals are made for each iteration of the algorithm; `criterion` specifies whether the threshold should be applied to the value of the likelihood (`value`), or to the values of the parameters (`variables`); `threshold` provides a stopping criterion: if an iteration finishes without there having been an increment in the likelihood (where `criterion` is `value`) or a variation in the MLE of the parameters (where `criterion` is `variables`) of at least threshold, it is assumed that the maximum has been found; `plot` has the same meaning as before.

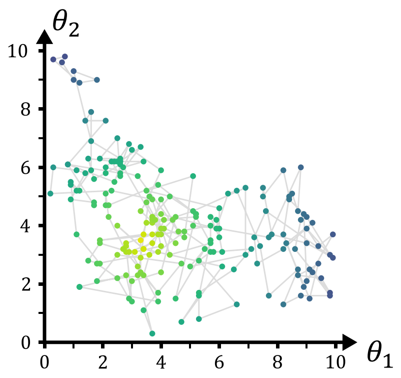


Figure 6. Values of $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$ sampled by a `Randomwalk(250, 0.01, value, plot)` strategy. The maximum-likelihood estimate is $\boldsymbol{\theta_1} = \mathbf{3.33}$ and $\boldsymbol{\theta_2} = \mathbf{3.45}$.

Figure 6 shows the values that would be sampled by a `Randomwalk(250,0.001,value,plot)` strategy when applied to the likelihood function of **Figure 3**; the maximum-likelihood estimates for the parameters θ_1 and θ_2 are 3.33 and 3.45, respectively, which is a noticeable improvement over the previous estimates (which were restricted to integer values only), even though it was necessary to compute 250 likelihoods to obtain them.

6.1.1.1.4 Nesterov climbing

Finally, under the "Nesterov climbing" approach, the likelihood function is optimised using gradient ascent with Nesterov acceleration (Nesterov, 2004). This approach leads to the closest local maximum and is therefore only useful when the initial "guess" for the parameter values is reasonably close to the ML values.

The syntax for this strategy is **`NesterovClimbing(steps,criterion,threshold,plot)`**, where the parameters have the same meaning as in the previous case. Figure 7 shows the values sampled by a `NesterovClimbing(50,0.001,value,plot)` strategy applied to the likelihood function of **Figure 3**; the ML estimates are very close to the true values, at $\theta_1 = 3.39$ and $\theta_2 = 3.38$, and 200 likelihoods had to be computed in order to obtain them (each step requires computation of the likelihood and of its gradient, which itself entails $n + 1$ likelihood computations, where n is the number of parameters to estimate).

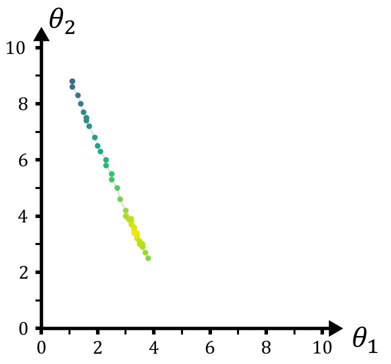


Figure 7. Values of $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$ sampled by a `NesterovClimbing(50, 0.01, value, plot)` strategy. The maximum-likelihood estimate is $\boldsymbol{\theta_1} = \mathbf{3.39}$ and $\boldsymbol{\theta_2} = \mathbf{3.38}$.

In sMap, it is also possible (and indeed, advisable) to combine multiple strategies by combining them with the `|` character, as in `Strategy1(parameters)|Strategy2(parameters)`. The strategies will be run one after the other, with the ML estimate from one strategy becoming the starting point for the next strategy. It is thus possible, for example, to obtain first a rough estimate using iterative sampling, then explore further the parameter space using random walk, and finally optimise the best estimate using Nesterov climbing.

Such a strategy, in this case, could be specified, for example, as:

```
IterativeSampling(1,10,1,0.001,plot)|RandomWalk(250,0.001,value,plot)| NesterovClimbing(50,0.001,value,plot)
```

The result of this strategy would be that the parameter values shown in **Figure 8** would be explored, and the final maximum-likelihood estimates would be very close to the true peak of the likelihood function.

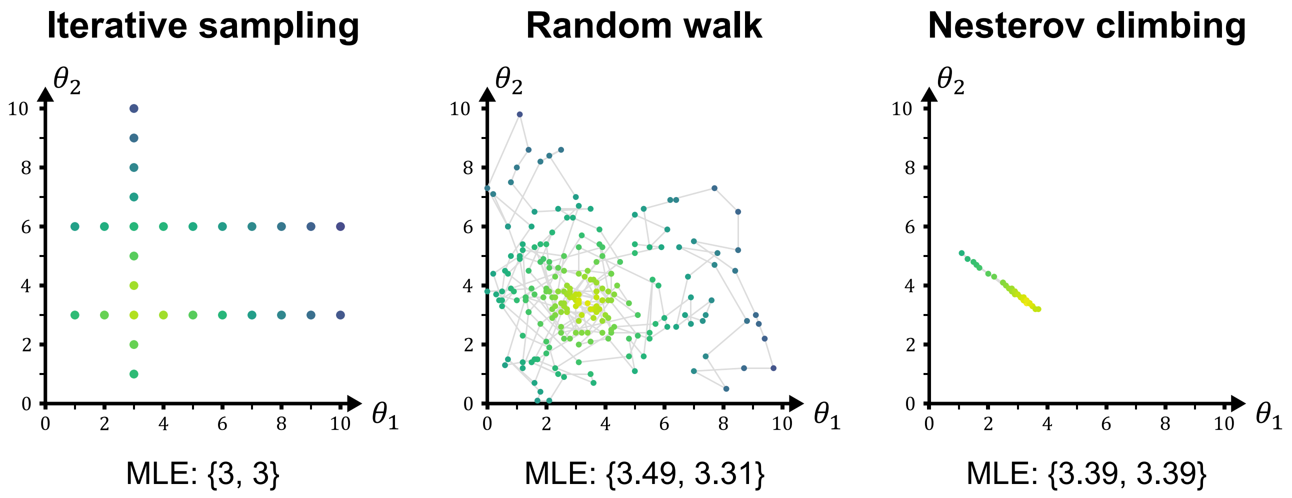


Figure 8. Steps in a combined maximum-likelihood optimisation strategy: first, a rough estimate is obtained using the iterative sampling strategy; this estimate is used as the starting point for a random walk, which explores more of the parameter space, and finally the maximum-likelihood estimate from this step is used as the starting point for a Nesterov climbing optimisation. The final estimate for the maximum likelihood coincides with the actual value.

The default maximum-likelihood optimisation strategy in sMap is, in fact a combined strategy: `IterativeSampling(0.0001,10.0001,0.1,plot,Value,0.001)|RandomWalk(Value,0.001,10000,plot)|NesterovClimbing(Value,0.001,100,plot)`

Since this (in particular, the first iterative sampling step) is optimised for rates between 0 and 10, it is advisable to "normalise" the length of the tree being analysed, so that the most likely values for the rates fall in this range. This can be done using the `-N/--norm` parameter of sMap (see 6.1.5).

6.1.1.2 Bayesian sampling

Another approach, instead of using the ML estimate or the parameters for every simulation, is to sample parameters according to their posterior probability distribution and use a different set of parameters for each simulation. This makes it possible to take into account uncertainty in the parameter specification, which will be reflected by an appropriate prior probability distribution. This prior distribution will be combined with the likelihood function to yield the posterior distribution.

6.1.1.2.1 Priors

In sMap, this approach is enabled by specifying priors for the parameters. The way priors are chosen is a subjective issue, therefore sMap implements many different priors from which a user can choose.

For root node priors, the only implemented priors are Dirichlet distributions. These are specified by supplying the Dirichlet parameter for each component (see 2.3.3).

For conditioned probabilities, it is possible to use Dirichlet priors, or multinomial (categorical) priors (which only allow the probabilities to be either 0 or 1, see 2.3.1).

For transition rates, there are 20 possible prior distributions, which are all the continuous univariate distributions implemented by the Math.Net Numerics library (https://numerics.mathdotnet.com/). Note that, since negative rates are not meaningful, all distributions are truncated at 0. Each distribution has some parameters, which are described in the following table:

Distribution	Description
`Beta(α,β)`	Beta distribution. For details, see https://en.wikipedia.org/wiki/Beta_distribution.
`BetaScaled(α,β,μ,σ)`	Scaled beta distribution: a beta distribution that has been translated and scaled in order to be nonzero between μ and μ + σ (instead of between 0 and 1).
`Cauchy(x₀,γ)`	Cauchy distribution. For details, see https://en.wikipedia.org/wiki/Cauchy_distribution.
`Chi(k)`	χ distribution with k degrees of freedom. For details, see https://en.wikipedia.org/wiki/Chi_distribution.
`ChiSquared(k)`	χ^2 distribution with k degrees of freedom. For details, see https://en.wikipedia.org/wiki/Chi-squared_distribution.
`ContinuousUniform(a,b)`	Uniform distribution defined between a and b. For details, see https://en.wikipedia.org/wiki/Uniform_distribution_(continuous).
`Erlang(k,λ)`	Erlang distribution. For details, see https://en.wikipedia.org/wiki/Erlang_distribution.
`Exponential(λ)`	Exponential distribution. For details, see https://en.wikipedia.org/wiki/Exponential_distribution.
`Fisher-Snedecor(d₁,d₂)`	Fisher-Snedecor distribution, also known as *F*-distribution. For details, see https://en.wikipedia.org/wiki/F-distribution.
`Gamma(α,β)`	Gamma distribution, parametrised with shape and rate. For details, see https://en.wikipedia.org/wiki/Gamma_distribution.
`InverseGamma(α,β)`	Inverse-gamma distribution. For details, see https://en.wikipedia.org/wiki/Inverse-gamma_distribution.
`Laplace(μ,b)`	Laplace distribution. For details, see https://en.wikipedia.org/wiki/Laplace_distribution.
`LogNormal(μ,σ)`	Log-normal distribution. For details, see https://en.wikipedia.org/wiki/Log-normal_distribution.
`Normal(μ,σ)`	Normal distribution. For details, see https://en.wikipedia.org/wiki/Normal_distribution.
`Pareto(xₘ,α)`	Pareto distribution. For details, see https://en.wikipedia.org/wiki/Pareto_distribution.
`Rayleigh(σ)`	Rayleigh distribution. For details, see https://en.wikipedia.org/wiki/Rayleigh_distribution.
`Stable(α,β,c,μ)`	Stable distribution. For more details, see https://en.wikipedia.org/wiki/Stable_distribution.
`StudentT(μ,σ,ν)`	Student's T-distribution. For more details, see https://en.wikipedia.org/wiki/Student%27s_t-distribution. Note that MathNet.Numerics implements a generalised version that also specifies the location (μ) and scale (σ). See Gelman *et al.*, 2013 for more details.
`Triangular(a,b,c)`	Triangular distribution. For more details, see https://en.wikipedia.org/wiki/Triangular_distribution.
`Weibull(k,λ)`	Weibull distribution. For more details, see https://en.wikipedia.org/wiki/Weibull_distribution.

Figure 9 shows the shapes of these prior distributions for some values of their parameters.

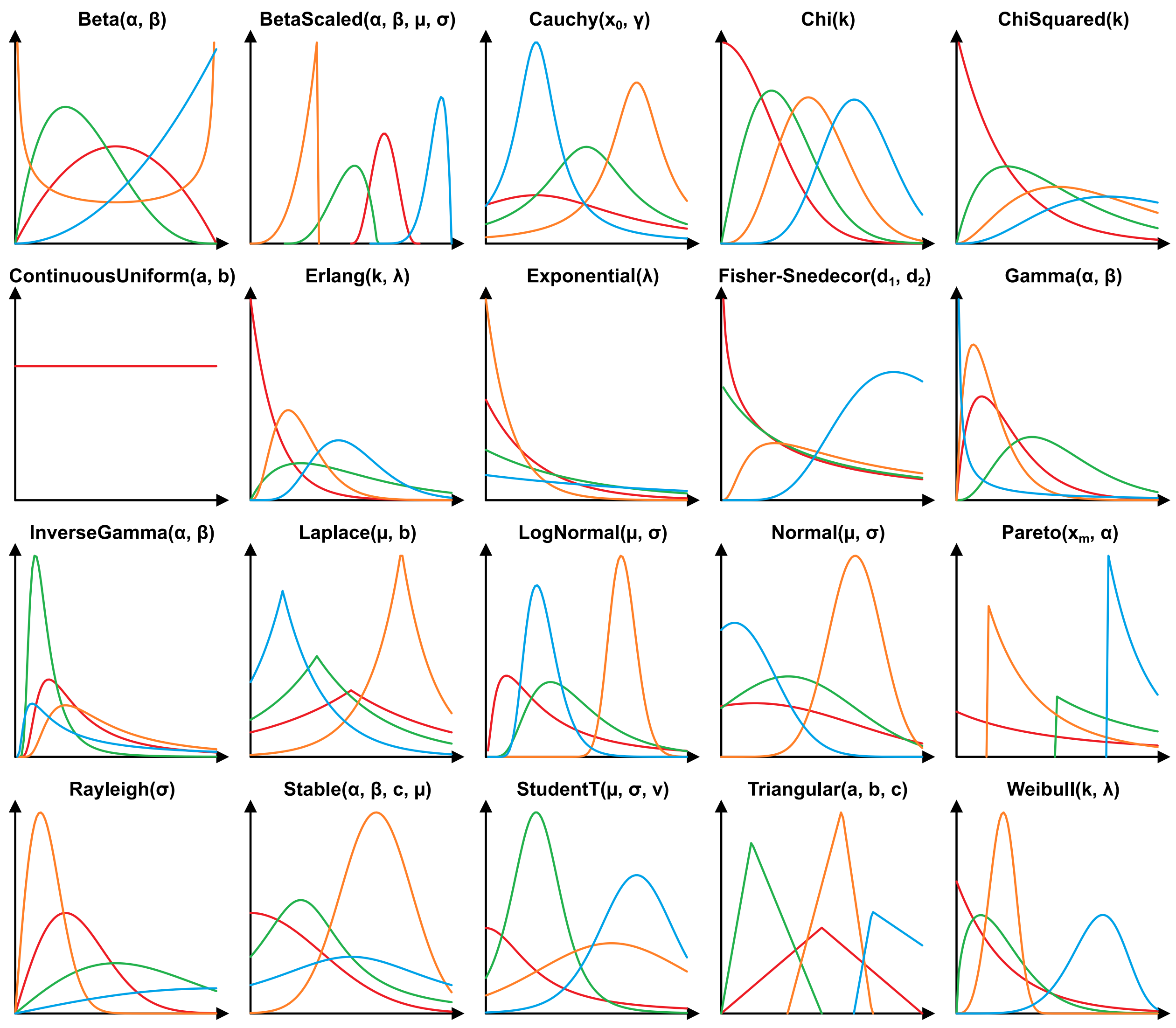


Figure 9. Prior distributions implemented in sMap. For each distribution, multiple examples (obtained using different values for the parameters of the distribution) are shown in different colours.

6.1.1.2.2 Sampling strategy

In a Bayesian analysis in sMap, parameters are sampled using a multithreaded Metropolis-coupled Markov chain Monte Carlo approach, with a mixed sampler: Gibbs sampling is used to sample the trees provided by the user (which means that it is assumed that the provided trees represent a posterior distribution), while random-walk Metropolis-Hastings sampling with normally distributed steps is used to sample the model parameters.

The proposal step sizes can be tuned for each parameter (using the `--estimate-steps` parameter, see 6.1.5), in order to achieve a target acceptance rate of the proposals (that can be provided using the `--acceptance-rate` parameter, see 6.1.5). Having appropriate step sizes is important to make sure that the chains are adequately "mixing", i.e. that they are exploring the posterior distribution as efficiently as possible. Step sizes that are too small cause the chain to only explore a very small range of parameter values, while excessively large step sizes cause the chain to be "stuck" in a local maximum because almost every proposal is rejected.

During the analysis, sMap will output various statistics that can be used to determine the status of the chains (i.e. whether they have converged or not). "Convergence tests" are implemented to automatically stop the analysis when the statistics reach particular thresholds. An example of these statistics is the following:

```
Performing MCMC sampling...

Sampling parameters for character 0
Performing initial burnin and determining step size: Done
Average estimated step size multipliers:
      0.501233303738212
      4.68585644780883
      1.11840796093434

   Sample    │ Acceptance │   Swap AR    │  MCoV Mean │   MCoV SD  │   Min ESS
─────────────┼────────────┼──────────────┼────────────┼────────────┼────────────
     100     │ 0.42817291 │     0.66     │ 0.05885473 │ 0.08213287 │     NC
     200     │ 0.4262022  │     0.59     │ 0.13094175 │ 0.06418351 │     NC
      …      │     …      │      …       │     …      │     …      │     …
    2200     │ 0.43159527 │ 0.40009477   │ 0.01210547 │ 0.04684493 │   523.4

Converged!
```

First, the step size multipliers that have been determined are shown (these can be supplied to the program in case it should be necessary to repeat the analysis, so that step size estimation can be skipped).

Then, a table is printed:

- The `Sample` column shows how many samples have been collected from the chain. 10% of these samples will be discarded as burn-in. For the chain to pass the "sample count" convergence test, an appropriate number of samples (which is provided by the `--min-samples` parameter, see 6.1.5) needs to have been collected. By default, this is twice as many as the simulations that will be run by the program. In this case, 1000 simulations have been requested, thus the required number of samples is 2000 (plus 10%, or 200 samples, for burn-in).
- The `Acceptance` column shows the overall rate of acceptance of the MCMC proposals. As has been said, this should be neither too high, nor too low, for the chain to have optimal performance.
- The `Swap AR` column shows the overall acceptance rate for chain swap proposals. This depends on the "temperature" of the chains (specified by the `--temp` parameter, see 6.1.5), and, again, should be neither too high, nor too low, for optimal performance.
- The `MCoV Mean` column shows the maximum coefficient of variation (CoV) for parameter means. The CoV for the mean of each parameter p is computed as follows:
  - For each run i, compute the mean value $\overline{p}_i$ of the parameter
  - Compute the mean and standard deviation of all the $\overline{p}_i$ (respectively, $\overline{\overline{p}}$ and $\sigma_{\overline{p}}$)
  - The CoV for p is $\frac{\sigma_{\overline{p}}}{\overline{\overline{p}}}$.

  The value shown in this column is thus the maximum $\max_p\left(\frac{\sigma_{\overline{p}}}{\overline{\overline{p}}}\right)$. When the runs have converged, they should be sampling the same distribution, thus this value should be as low as possible. When it goes below the threshold defined by the `--max-cov` parameter (see 6.1.5), the "mean CoV" convergence test has passed.
- The `MCoV SD` column shows the maximum CoV for parameter standard deviations. It is computed in a similar way to the maximum CoV of the mean (only, instead of considering mean values of the parameters, the standard deviations σ_{p_i} are used). Again, when the runs have converged this value should be close to 0, and when it goes below the threshold defined by the `--max-cov` parameter (see 6.1.5), the "standard deviation CoV" convergence test has passed.

- The `Min ESS` column shows the minimum effective sample size (ESS) for each parameter in each run. An ESS is computed separately for the parameters, and then the minimum value is reported. When this value is greater than the threshold specified by the `--min-ess` parameter (see 6.1.5), the "ESS" convergence test has passed. Note that, since determining the ESS is a computationally expensive process, ESS are computed only when all the other convergence tests have passed.

When all the convergence tests (sample count, mean CoV, standard deviation CoV and ESS) have passed, the analysis is automatically stopped, and the parameter samples used for the stochastic mapping simulations. However, it is also possible to manually stop the analysis by pressing CTRL+C, which will cause sMap to behave as if all the tests had passed. When this is not possible (e.g. when the program is run in a noninteractive environment such as a cluster), the `--poll` option (see 6.1.5) provides an alternative way to stop the analysis by creating a specific file.

In any case, at the end of the Bayesian analysis an adequate number of samples will have been collected for each parameter.

6.1.2 Ancestral state reconstruction

For each set of parameter samples, a Bayesian ancestral state reconstruction is performed. To do this, likelihoods are computed using Felsenstein's "pruning" algorithm (Felsenstein, 1973, 1981) for each character state and for each node in the tree.

Then, priors and posteriors are computed in a similar way, following the algorithm described by (Nielsen, 2002), i.e. climbing the tree from the root: the root node priors are given by the user and are used in conjunction with the transition rate matrix to obtain the probability distributions for the rest of the nodes that are used to sample the state of each node.

The end result is that for each set of parameter samples, we have a posterior sample for the state of the characters at each node of the tree.

6.1.3 Simulation

To obtain complete character history realisations, the evolution of the characters along the trees obtained in the previous step is simulated, using the model parameters.

For an independent character, the history is simulated branch-by-branch using the rate matrix to determine when state changes occur. If at the end of the simulation for a branch the state does not correspond to the one that has been sampled, the simulated branch is rejected, and a new simulation is run. The process is the same for dependent characters, which are joined in a single character which behaves as an independent character.

For conditioned characters, first of all the history of each conditioning character is simulated (always working branch-by-branch). Then, the history for the conditioning character is simulated, by sampling it only when a conditioning character changes. At the end of the branch, if the state does not correspond to the sampled state, all the simulations for the conditioned and conditioning characters are rejected.

At the end of this step the result will be a number of simulated histories, each describing one of the many ways the characters might have evolved along a tree.

6.1.4 Summarisation of results

To summarise the character histories, it is necessary to specify a single tree that will be used as the basis for this process. If every character simulation has been run using the same tree, then this same tree should be used as the "summary tree", otherwise a consensus tree of some kind should be chosen.

For each point of each branch in the tree (including the nodes, i.e. the terminal points of the branches), the posterior probability that a character c is in a state i at that time and along that branch is determined by first of all considering only those simulations whose tree actually has that branch at that point in time. Then, amongst these simulations, what proportion has character c in state i at that point is determined, which represents the posterior probability.

The result of the stochastic mapping analysis is thus a plot that tells us the probability that a character was in a certain state not only on every node in the tree, but also at every intermediate point in time.

6.1.5 Command-line parameters

The sMap program has multiple command-line parameters, which are used to provide data and to change how the algorithms work.

Here is a description of the command line parameters:

Parameter	Description
`-h` `--help`	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-t <treefile>`
`--tree <treefile>`
	Required. Specifies the file containing the trees that will be used for the stochastic mapping analysis, in Newick format (see 2.1).

`-d <datafile>`
`-x <datafile>`
`--data <datafile>`
	Required. Specifies the data file, containing character states in a relaxed PHYLIP format (see 2.2).

`-o <outputprefix>`
`--output <outputprefix>`
	Required. Specifies the location and name of the output files, which will be called `<outputprefix>.something`. The output prefix can include a (relative or absolute) path to a different folder (but the folder must exist, otherwise the program will exit with an error).

`-n <numsim>`
`--num-sim <numsim>`
	Required. Specifies the number of stochastic mapping simulations to run. This is also the number of samples drawn from the posterior distribution for Bayesian analyses. `<numsim>` must be greater than 0, and suggested values are in the order of 1000.

`-T <treefile>`
`--mean-tree <treefile>`
	Optional/Required. Specifies a "mean tree" that is used to summarise the analysis. If the tree file for the -t/--tree parameter contains multiple trees, this parameter is required. If it contains only one tree, this parameter is optional, and its default value is equal to that of the -t/--tree parameter.

`-a <archivefile>`
`--archive <archivefile>`
	Optional. Specifies an ZIP compressed file which contains a file named `.args` with the command line arguments to use for the analysis. The file can/should also contain other files (e.g. data files for the analysis), which should be referenced in the `.args` file. The ZIP file will be extracted to a temporary folder and the working directory of the application will be set to that folder prior to the parsing of the arguments from the `.args` file. If this argument is specified, the only other argument allowed is `-o/--output` (which is required).

`-D <dependencyfile>`
`--dependency <dependencyfile>`
	Optional. Specifies a NEXUS file containing "Dependency" blocks that specify relationships between characters (see 2.3.1). Characters whose relationships have not been specified are assumed to be independent.

`-r <ratefile>`
`--rates <ratefile>`
	Optional. Specifies a NEXUS file containing “Rates” blocks that specify transition rates between character states (see 2.3.2). Rates that are not specified are estimated by maximum-likelihood.

`-p <pifile>`
`--pi <pifile>`
	Optional. Specifies a NEXUS file containing “Pi” blocks that specify root node priors for the character states (see 2.3.3). Priors that are not specified are assumed to be equal to $^1/_n$, where n is the number of states.

`-i <modelfile>`
`--input <modelfile>`
	Optional. Specifies a NEXUS file containing “Dependency”, “Rates” or “Pi” blocks. This is the same as providing the same file to the `-D/--dependency`, `-r/--rates` and `-p/--pi` parameters.

`-s <seed>`
`--seed <seed>`
	Optional. Specifies a random number seed. If this is provided, thread-local random number generators (RNG) will be initialised with seeds extracted from an application-global RNG that has been initialised with the provided seed. Otherwise, the thread-local RNG will be initialised using seeds drawn from an application-global cryptographically secure RNG. Using this parameter is discouraged except in debug settings, as it may degrade the quality of the generated random numbers. **Default**: none.

`-N [factor]`
`--norm [factor]`
	Optional. If this parameter is specified, the branch lengths of the tree are normalised by dividing them by `factor`. If factor is not provided, it is taken to be equal to the height of the mean tree provided to the `-T/--mean-tree` parameter. **Default**: disabled.

`-c <threshold>`
`--coerce <threshold>`
	Optional. If this option is enabled, any branch length smaller than `threshold` (after normalisation, if applicable) is set to be equal to `threshold`. This can be useful e.g. to enforce branch lengths that are strictly positive. **Default**: disabled.

`-m <strategy>`	**Optional**. Maximum-likelihood optimisation strategy (in the format described in 6.1.1.1). It can be a combination of multiple strategies, which are executed in sequence. Note that to prevent command-line misinterpretations, it is often required to put the strategy between quotation marks (“”). **Default**: `IterativeSampling(0.0001,10.0001,0.1,plot,Value,0.001)|RandomWalk(Value,0.001,10000,plot)|NesterovClimbing(Value,0.001,100,plot).`

`--nt <numthreads>`
`--num-threads <numthreads>`
	Optional. Specifies the number of threads to use in the prior and posterior computation step, as well as the maximum number of threads for the simulation step (for this step, the runtime will determine the actual number of threads, up to `<numthreads>`). Note that this does not affect the number of threads used in the MCMC sampling steps. **Default**: 1.

`--dt <numsamples>`
`--d-test <numsamples>`
	Optional. Perform a D-test for correlation between different characters, sampling `<numsamples>` histories from the posterior predictive distribution of each parameter sample. Note that will cause `<numsamples>` * `<numsim>` histories to be sampled, which may take some time. **Default**: 0 (disabled).

`--pp <numsamples>`
`--posterior-predictive <numsamples>`
	Optional. Sample `<numsamples>` histories from the posterior predictive distribution of each parameter sample. These may be later used with Stat-sMap to perform D-tests. Note that will cause `<numsamples>` * `<numsim>` histories to be sampled and saved, which may take some time and will increase the output file size. **Default**: 0 (disabled).

`--poll`	**Optional**. If enabled, during MCMC sampling the program will periodically look for a file called `<outputprefix>.interrupt`. If such a file is found, the MCMC analysis will be assumed to have converged as if a CTRL+C had been detected (and the file will be deleted). This is useful when running sMap in a non-interactive environment (such as a cluster), where it is not feasible to manually send CTRL+C signals to the program. **Default**: disabled.
`--num-runs <numruns>`	**Optional**. Number of runs independent parallel runs to use in a MCMC analysis. To enable most convergence statistics, this must be at least 2. **Default**: 2.
`--num-chains <numchains>`	**Optional**. Number of parallel Metropolis-coupled MCMC chains. Increasing this number slows down each step of the analysis, but allows the chain to more easily explore the whole distribution. **Default**: 4.
`--temp <coefficient>`	**Optional**. Coefficient used to the determine the “temperature” of the MCMC chains. The temperature of chain i is determined as $(1 + i \cdot \text{coefficient})^{-1}$. Higher values for this parameter allow the heated chains to explore the distribution more quickly, at the expense of reducing the acceptance rate of chain swap proposals. **Default**: 0.5.

`--sf <n>`
`--sampling-frequency <n>`
	Optional. Determines how often the MCMC chains are sampled (i.e. every `<n>` generations). Increasing this will reduce autocorrelation between samples (and thus increase ESS, the same number of samples), but will increase how long it takes to draw each sample. **Default**: 10.

`--wf <n>`
`--swap-frequency <n>`
	Optional. Determined how often a chain swap proposal is made. Must be a multiple of the sampling frequency. **Default**: 10.

`--df <n>`
`--diagnostic-frequency <n>`
	Optional. Determines how often diagnostic statistics are computed and printed. If this is too low, the program will spend much of its time computing these statistics, rather than doing useful computations. If it is too high, it will become hard to assess the status of the analysis. **Default**: 1000.

`--min-samples <n>`	**Optional**. Minimum number of samples that need to be collected from the chain for the analysis to pass the convergence test. **Default**: $2 \cdot$ `<numsim>`.
`--max-cov <value>`	**Optional**. The coefficient of variation for the mean and standard deviation of each parameter that is being sampled must be lower than `<value>` for the analysis to pass the convergence test. **Default**: $$-\frac{1}{16}\ln\left(1-\frac{1}{3}\left(\frac{7}{3}-\frac{2}{2\cdot < \text{numruns} > -1}\right)\right)$$

`--min-ess <value>`	**Optional**. Minimum effective sample size that each parameter in each run must have for the analysis to pass the convergence test. **Default**: 200.
`--estimate-steps`	**Optional**. If enabled, optimal MCMC proposal step sizes will be estimated for each parameter. If disabled, the default step sizes will be used (but the initial burn-in phase will be faster). **Default**: enabled.
`--tuning-attempts <value>`	**Optional**. Number of tuning attempts to estimate the optimal MCMC proposal step sizes. Increasing this number will provide (slightly) more accurate estimates for the step sizes, but it will increase the duration of the initial burn-in phase for each MCMC analysis. Lowering this value will provide worse estimates for the step sizes, but will cause the initial burn-in phase to be shorter. **Default**: 10.
`--tuning-steps <value>`	**Optional**. Number of MCMC steps in each tuning attempt to estimate the optimal MCMC proposal step sizes. Increasing this number will provide (slightly) more accurate estimates for the step sizes, but it will increase the duration of the initial burn-in phase for each MCMC analysis. Lowering this value will provide worse estimates for the step sizes, but will cause the initial burn-in phase to be shorter. **Default**: 100.
`--acceptance-rate <value>`	**Optional**. Target “magic” acceptance rate for a single chain to aim for during step size estimation. If multiple chains are used (`<numchains>` > 1), the actual acceptance rate will be higher. Both too low and too high acceptance rate values cause the chain to be stuck and not adequately explore the target distribution (for different reasons). **Default**: 0.37.

`--sm <values>`
`--step-multipliers <values>`
	Optional. MCMC proposal step size multipliers. If only one values is specified, it is applied to all variables. If multiple values are specified (separated by a comma, e.g. `1,2,3,4`) each will be applied to a variable. Note that if –`-estimate-steps` is enabled (which is the default), this only has the effect of shifting the initial estimate point. **Default**: 1.

`--prior`	**Optional**. If enabled, all likelihood computations will be disabled during the MCMC analysis, and the parameters will be sampled according to their prior distribution. **Default**: disabled.
`--ss` `--stepping-stone`	**Optional**. If enabled, a stepping-stone analysis will be performed after the parameters for the analysis have been sampled, in order to estimate the marginal likelihood of the model. **Default**: disabled.
`--ss-steps <number>`	**Optional**. Number of steps in the stepping-stone analysis. Higher numbers result in more accurate estimates for the marginal likelihood, but cause the analysis to take linearly longer to execute. **Default**: 8.
`--ss-shape <value>`	**Optional**. Shape parameter of the beta distribution determining the likelihood exponent for each step in a stepping-stone analysis. For an analysis of this parameter, see Xie *et al.*, 2011. **Default**: 0.3.
`--ss-samples <number>`	**Optional**. Number of MCMC samples to gather for each step in a stepping-stone analysis. **Default**: `<numsim>`.

`--ss-estimate-steps`	**Optional**. If enabled, optimal MCMC proposal step sizes will be estimated for each step in a stepping-stone analysis. If disabled, the step sizes determined for the first MCMC run (that samples the posterior distribution) will be used. Enabling this option causes the initial burn-in phase for each step to take longer, but the chains converge faster. **Default**: disabled.
`--parameters <paramfile>`	**Optional**. A comma-separated list of files (one for each set of characters), containing samples for every parameter in the model. Each file can have header rows, and must contain exactly `<numsim>` data rows, each with a single value for each parameter in the model. This makes it possible to re-use parameter distributions computed in a previous sMap analysis (or by any other means).

`--pw <width>`
`--plot-width <width>`
	Optional. Page width (in points) for the PDF plots. **Default**: 500.

`--ph <height>`
`--plot-height <height>`
	Optional. Page height (in points) for the PDF plots. If not specified, it will be determined automatically for each plot. **Default**: auto.

`--br <rule>`
`--bin-rule <rule>`
	Optional. Rule to determine the number of bins (k) or bin width (w) in histogram plots, based on the number of samples n. **Possible values** (for more details see https://en.wikipedia.org/wiki/Histogram#Number_of_bins_and_width):
	• *Sqrt*: $k = \sqrt{n}$
	• *Sturges*: $k = \log_2 n + 1$
	• *Rice*: $k = 2 \cdot \sqrt[3]{n}$
	• *Doane*: $k = 1 + \log_2 n + \log_2 \left(1 + \frac{\lvert g_1 \rvert}{\sigma_{g_1}}\right)$
	• *Scott*: $w = \frac{7}{2} \cdot \frac{\sigma}{\sqrt[3]{n}}$
	• *FreedmanDiaconis*: $w = 2 \cdot \frac{\mathrm{IQR}}{\sqrt[3]{n}}$
	Default: FreedmanDiaconis.

6.2 ChainMonitor

The ChainMonitor program included in sMap can be used to examine trace files to determine whether a Bayesian analysis has converged, in a manner similar to the program Tracer (Rambaut *et al.*, 2018). When supplied with one or more trace file, the program will compute means and standard deviations for every parameter that has been sampled, as well as display a plot of the distribution of each parameter in each trace file.

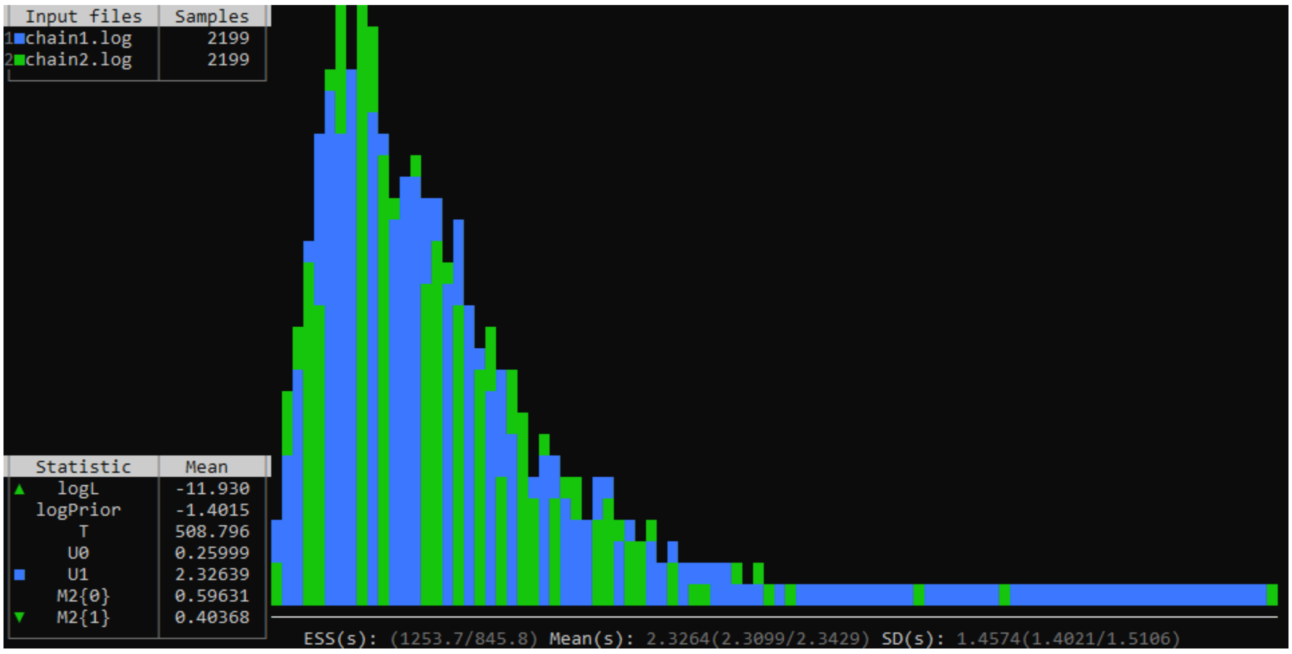
This program can be invoked as:

```
ChainMonitor [options] <file1> [<file2> […]]
```

In a UNIX environment, it is possible to use wildcards in the file name (e.g. `output.run*`); this will not work on Windows, however.

6.2.1 Interface

An example of the interface of ChainMonitor is the following:



```
 Input files  Samples
1■chain1.log     2199
2■chain2.log     2199

 Statistic     Mean
▲   logL      -11.930
  logPrior    -1.4015
     T        508.796
     U0       0.25999
■    U1       2.32639
   M2{0}      0.59631
▼  M2{1}      0.40368

ESS(s): (1253.7/845.8) Mean(s): 2.3264(2.3099/2.3429) SD(s): 1.4574(1.4021/1.5106)
```

This shows that the program is being used to analyse two files called `chain1.log` and `chain2.log`, each of which contains 2199 samples. The plot currently shows the distribution for parameter `U1` (what these parameter names mean depends on the program that has been used to run the analysis; for sMap this is described in the `<output>.paramNames.txt` output file).

The plot is effectively an overlay of histograms of the sampled distributions. Each trace file is plotted in the colour indicated to the left of the file name (in the top left corner). For each column of the plot, the trace file which has less samples in that bin is plotted "in front of" the other (for example, for the first column to the left of this plot, the `chain1.log` file had twice as many samples falling in that bin than the `chain2.log` file).

The bottom row shows some statistics for the sampled parameter distribution:

- `ESS`: effective sample size, computed for each trace file separately.
- `Mean`: the number outside the brackets is the overall mean of the sampled values, while the ones inside are the means computed for each trace file separately.
- `SD`: similarly, the number outside the brackets is the overall standard deviation of the sampled values, while the numbers inside are computed for each trace file separately.

Convergence can be assessed by looking at these statistics (the ESS should be as high as possible, at least greater than 200; all the means and standard deviations should be close to each other) and at the plot: in a converged analysis, the colours of the columns should appear rather randomly assorted (as in this example), which means that the sampled distributions are very close to each other.

6.2.2 Navigation

This interface can be navigated using the keyboard as follows:

- Pressing 1, 2, …, 9, 0 (depending on how many trace files have been opened) enables or disables the corresponding trace file (e.g. in this example, pressing 1 would remove the blue plot, only leaving the green one). Note that at least one file must be enabled at all times.
- The up and down arrows change the parameter that is displayed (highlighted by the blue square in the bottom left).
- When the text that should appear in the bottom row is longer than the available window width, the left and right arrows are used to scroll this text.

- Pressing the Q key closes the program.

6.2.3 Command-line parameters

The ChainMonitor program **requires** the names of the files to analyse as arguments; other command-line parameters are:

Parameter	Description
`-h` `--help`	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-b <burn-in>`
`--burn-in <burn-in>`
	Optional. Specifies the number of samples to discard as initial burn-in. This can be either a number (e.g. 1000), or a percentage of the number of samples for each file (e.g. 25%). **Default**: 10%.

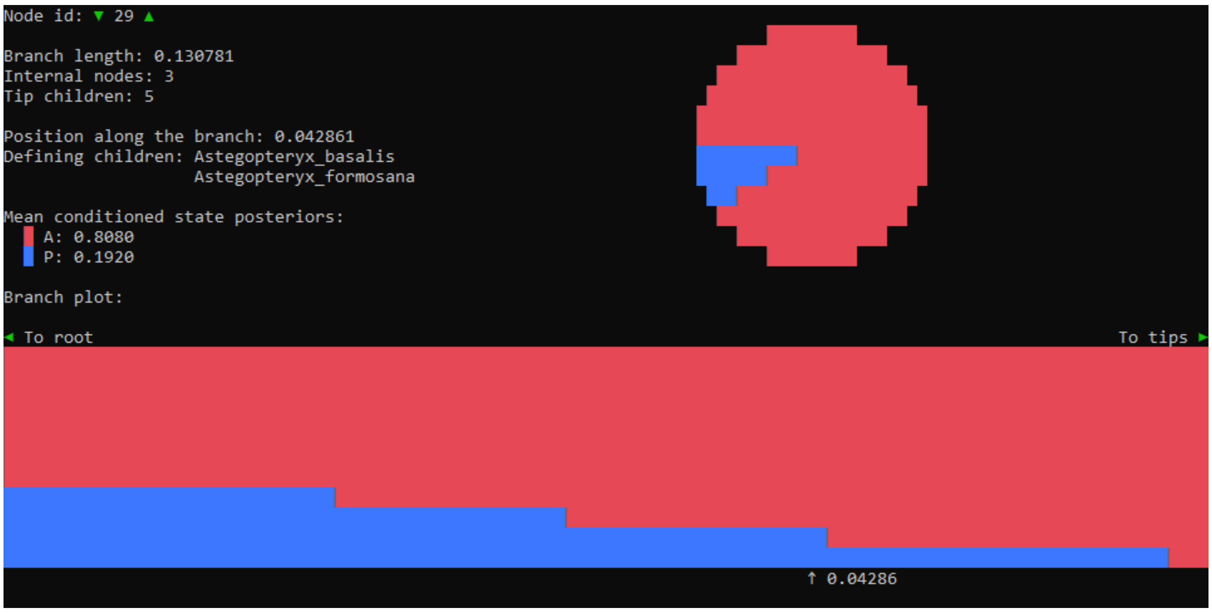
6.3 NodeInfo

The NodeInfo program can be used to obtain quantitative information about the state of nodes in a sMap analysis.

NodeInfo reads the `<output>.smap.bin` files produced by sMap and displays the information that has been saved therein; it can be run either in interactive mode, or in batch mode (by using the appropriate command line parameters).

6.3.1 Interface

An example of the interface of NodeInfo running in interactive mode is the following:



```
Node id: ▼ 29 ▲

Branch length: 0.130781
Internal nodes: 3
Tip children: 5

Position along the branch: 0.042861
Defining children: Astegopteryx_basalis
                   Astegopteryx_formosana

Mean conditioned state posteriors:
  🟥 A: 0.8080
  🟦 P: 0.1920

Branch plot:

◄ To root                                                                                      To tips ►
                                                                      ↑ 0.04286
```

The top left shows the ID of the node of the tree for which data is being shown.

Below that, the length of the branch ending in that node, the number of internal nodes that descend from the selected node and the number of leaves are shown.

Then, the point along the branch for which the probabilities are being displayed is shown. This is expressed in the same units as the branch length; a value of 0 means that the probabilities for the node itself are being displayed, while greater values are measured along the branch, going towards the node's parent.

The `Defining children` are any two tips of the tree whose last common ancestor is the node that is being analysed. When a terminal node is selected, the name of the taxon corresponding to that node is shown instead.

Following this, the posterior probabilities that the branch is in each state at the chosen point are shown.

The "pie chart" in the top right shows these probabilities for the node itself, while the char at the bottom shows them for the whole branch (in scale).

The bottom row shows the current position across the branch.

6.3.2 Navigation

The interface can be navigated with the keyboard:

- The up and down arrows change the node currently being displayed.
- The left and right arrows change the position along the branch.
- Pressing the Q key closes the program.

6.3.3 Command-line parameters

NodeInfo has multiple command-line parameters, which can be used to select which node to analyse, or to skip the interactive session and just print the statistics to the console.

Parameter	Description
`-h` `--help`	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-s <file>`
`--smap <file>`
	Required. Specifies the input sMap file. This should be a `sMap.bin` file produced by the sMap program.

`-n <id>`
`--node-id <id>`
	Optional/Required. Specifies which node(s) to analyse using node ids. If the program is not run in interactive mode (see `--simple` and `--batch` below), multiple ids can be supplied, separated by commas (e.g. `1,2,3`). At least one between `-n`/`--node-id` and `-m`/`--mono-constraints` must be supplied.

`-m <t1> <t2>`
`--mono-constraint <t1> <t2>`
	Optional/Required. Specifies which node to analyse as the last common ancestor of `t1` and `t2`. If the program is not run in interactive mode (see `--simple` and `--batch` below), this option can be specified multiple times. At least one between `-n`/`--node-id` and `-m`/`--mono-constraints` must be supplied. If the completion scripts have been installed (on a UNIX platform, using the bash shell), it is possible to use tab-completion with this parameter.

`--simple`	**Optional**. Display node information without plot and exit. The same information that would be displayed in interactive mode (except for the plots) is printed. **Default**: disabled.
`--batch`	**Optional**. Only display specific information (see the -f/--format option below) and exit. By default, displays node ids and state probabilities. **Default**: disabled.

`--f <format-string>`
`--format <format-string>`
	Optional. When combined with the --batch option (see above), specifies the information to be included in the output string. Escape characters can be used (e.g. \t for tabulation and \n for newline). Available output parameters are:
	• `%I%`: node id
	• `%P%`: position along the branch
	• `%L%`: branch length
	• `%C%`: number of internal children

	• `%l%`: number of leaves • `%D1%`: defining child 1 • `%D2%`: defining child 2 • `%##%`: total number of states • `%#n%`: name of state `n` (where `n` is a 0-based integer). For example, to get the name for the second state use `%#1%`. • `%p(#n)%`: posterior probability of state `n` (where `n` is a 0-based integer). For example, to get the posterior probability of the first state, use `%p(#0)%`. • `%p(s)%`: posterior probability of state s (where s is the name of the state). For example, to get the posterior probability of state `A`, use `%p(A)%`.

6.4 Blend-sMap

The Blend-sMap program is used to “blend” multiple stochastic mapping analyses, giving appropriate weight to each one; this can be useful, for example, to average multiple analyses using the posterior probabilities of the models that have been used as weights (see 5.6 for an example).

A new sMap output file is produced, containing the requested number of simulations that have been extracted from the input files, proportionally to the specified weights.

6.4.1 Command-line parameters

The command-line parameters of Blend-sMap are used to specify the input and output files, the weight of each input file and the number of simulations to include in the output file.

Parameter	Description
-h **--help**	**Optional**. Shows the help message, with a description of the command line parameters, and exits.
-s <file>,<weight> **--smap <file>,<weight>**	**Required**. Specifies an input sMap file and its weight (separated by a comma). The file should be a `sMap.bin` file produced by the sMap program; the weight can be a number in any scale (all weights are normalised by the program). For example, `smap.bin,0.5` and `smap.bin,50` are both valid. This parameter should be specified once per input file (with the appropriate weight).
-n <numsim> **--num-sim <numsim>**	**Required**. Number of simulated histories to include in the output file. These will be drawn from the input files proportionally to the weights.
-o <outputfile> **--output <outputfile>**	**Required**. Specifies the name of the output file.

6.5 Merge-sMap

The Merge-sMap program is used to “merge” multiple stochastic mapping analyses that have been run using different data files (see 5.11 for an example), or to extract the data about specific characters from a sMap run file containing data for multiple characters. It can also be used to rename the character states of an analysis that has already been performed.

A new sMap output file is produced, containing the requested number of simulations that have been extracted from the input files. It is also possible to use this program to export the simulations in a format that is compatible with the *phytools* R package.

6.5.1 Command-line parameters

The command-line parameters of Merge-sMap are used to specify the input and output files, the characters to use from each input file and the number of simulations to include in the output file.

Parameter	Description
`-h` `--help`	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-s <file>;<char1>[,<char2>...]`
`--smap <file>;<char1>[,<char2>...]`
	Required. Specifies an input sMap file and the characters to extract from that file. The file should be a `sMap.bin` file produced by the sMap program; multiple characters can be specified by separating them with commas. For example, `smap.bin;0,1` refers to characters 0 and 1 from the file `smap.bin`. This parameter should be specified once per input file (with the characters).

`-n <numsim>`
`--num-sim <numsim>`
	Required. Number of simulated histories to include in the output file. These will be drawn from the input files for the requested characters.

`-r [<name_file>]`	**Optional**. Renames the character states, using the entries provided in the `name_file` (or, if not provided, numbers starting with 0). This makes it also possible to "merge" multiple characters into a single character, or to "split" a single character in multiple sub-characters. The `name_file` should contain one entry per line, where each entry consists of the original state name, a white-space (or tabulation) and the new state name. Example:
	`0,0     A`
	`0,1     B`
	`1,0     C`
	`1,1     D`

`-o <outputsmap>`
`--output-smap <outputsmap>`
	Optional. Specifies the name of the sMap output file. At least one between this option and the next one should be specified.

`-p <outputsmap>`
`--output-phytools <outputphytools>`
	Optional. Specifies the name of the phytools output file. At least one between this option and the next one should be specified.

6.6 Stat-sMap

The Stat-sMap program is used to perform computations on stochastic mapping.

It can be used to compute the number of transitions, the time spent in each character state, as well as to perform D-tests (Huelsenbeck, Nielsen and Bollback, 2003, see 5.11 for an example).

By default, this program will print the number of transitions and the time spent in each state. If the sMap file contains histories for multiple characters, the transitions and time will refer to the combined state (e.g. for two characters, each with states A and B, the transitions it will report will be $A, A \rightarrow A, B$, or $A, A \rightarrow B, A$ etc. and the times will be the time spent in A, A, in A, B etc.).

If the `-m/--marginal` switch is specified, the time spent and transitions will be summed for each character (e.g. for two characters, each with states A and B, the transitions it will report will be $A \rightarrow B$

and $B \rightarrow A$ for the first character, and $A \rightarrow B$ and $B \rightarrow A$ for the second character, while the times will be the time spent in A or B for the first character, and in A or B for the second character).

If the `-t`/`--d-test` parameter is used, a D-test will be perform between the specified characters. The results of the D-test will be printed on screen, as well as saved in a PDF file (which will include the posterior-predictive distribution of the D-statistic, plotted as a histogram and as a continuous kernel density estimate approximation).

6.6.1 Command-line parameters

The command-line parameters of Stat-sMap are used to specify the input file and the analysis to perform.

Parameter	Description
`-h` `--help`	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-s <file>`
`--smap <file>`
	Required. Specifies the input sMap file.

`-m` `--marginal`	**Optional**. If this parameter is specified, the times and transitions specified will be marginalised over each individual character.

`-o <outputsmap>`
`--output-smap <outputsmap>`
	Optional. Specifies the name of the sMap output file. At least one between this option and the next one should be specified.

`-t <char1> <char2> <output>`
`--d-test <char1> <char2> <output>`
	Optional. If this parameter is specified, a D-test between characters `<char1>` and `<char2>` will be performed. An output file called `<output>.pdf` will be created.

6.7 Plot-sMap

The Plot-sMap program is used to produce plots of sMap analyses. Its interface can be used to change various parameters of the plot and to produce different kinds of plots.

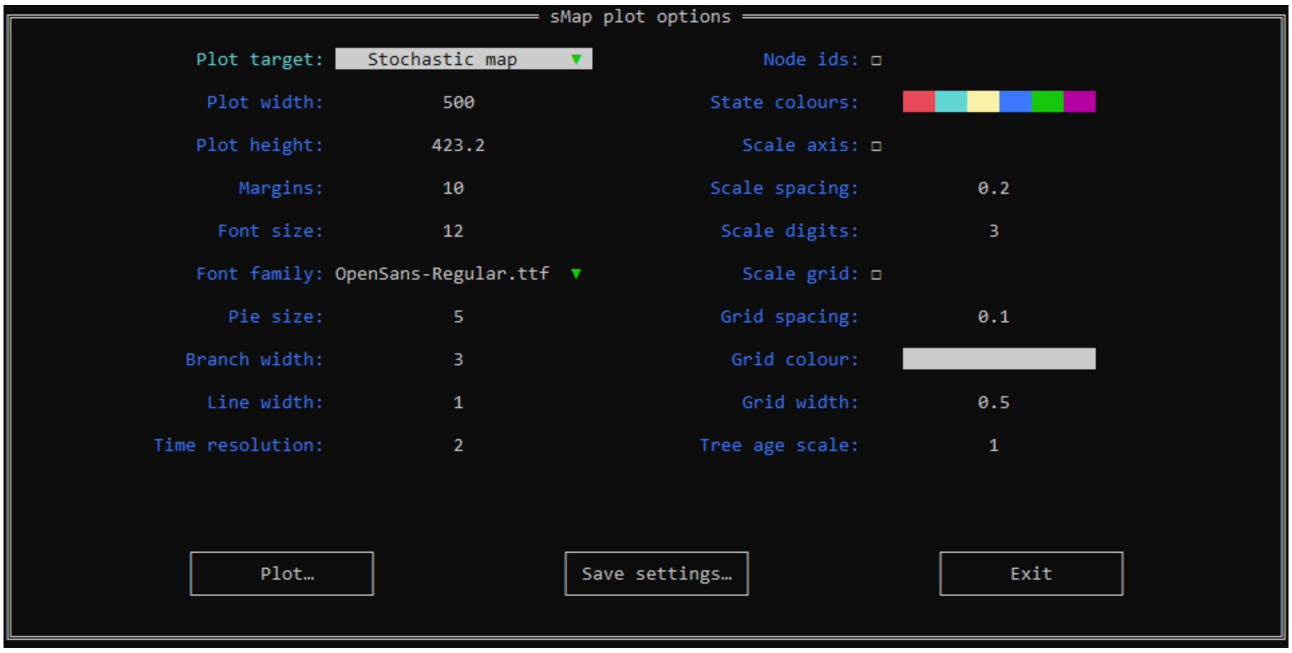
Since this is a command-line program, it cannot provide a real-time preview of the output. Therefore, it is suggested that you open the PDF plot produced by Plot-sMap using a PDF viewer that does not lock the file (unfortunately, if you open the PDF in Adobe Reader, it will prevent Plot-sMap from updating it until you close the file). On Windows, you can use for example the Sumatra PDF reader (https://www.sumatrapdfreader.org/free-pdf-reader.html), which will automatically update the PDF display when the file is updated. On Linux you can use e.g. Evince (http://projects.gnome.org/evince/), while on macOS you can use the included Preview program.

Plot-sMap can be run either in interactive mode, or in batch mode (by using the appropriate command-line parameters). When run in batch mode, the program will perform the requested task and exit.

6.7.1 Interface and navigation

Plot-sMap has a main interface (that is first displayed when the program is opened), on top of which various dialogs can open to request information from the user. The typical case is when the value of a field needs to be changed: a list of the possible values will open.

An example of the interface of Plot-sMap is the following:

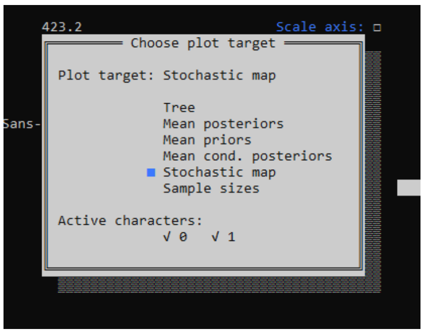


```
                              sMap plot options

      Plot target:  Stochastic map    ▼            Node ids: □
       Plot width:        500                 State colours: 🟥🟦🟨🟦🟩🟪
      Plot height:       423.2                   Scale axis: □
          Margins:        10                  Scale spacing:        0.2
        Font size:        12                   Scale digits:         3
      Font family: OpenSans-Regular.ttf  ▼       Scale grid: □
         Pie size:         5                   Grid spacing:        0.1
     Branch width:         3                    Grid colour:
       Line width:         1                     Grid width:        0.5
  Time resolution:         2                 Tree age scale:         1


         Plot…                Save settings…               Exit
```

The up, down, left and right arrows on the keyboard can be used to highlight the various options of the program. The value of each field can be changed by highlighting it and pressing enter; for numeric fields (e.g. `Plot width` or `Tree age scale`), this enables editing of the number, which can be done normally with the keyboard; pressing enter saves the new value. For some fields (detailed below), pressing enter will open a dialog to request more information. For checkbox fields (i.e. `Node ids`, `Scale axis` and `Scale grid`), enter toggles the state of the checkbox.

6.7.1.1 Plot target

The `Plot target` field determines what kind of plot is produced. When the value of this field is edited, a dialog with the list of all available plots opens (shown on the right).



```
423.2                                Scale axis: □
      ═══════ Choose plot target ═══════
      Plot target: Stochastic map

                   Tree
                   Mean posteriors
Sans-              Mean priors
                   Mean cond. posteriors
                 ■ Stochastic map
                   Sample sizes

      Active characters:
                   √ 0   √ 1
```

- When `Tree` is selected, the plot consists only of the phylogenetic tree underlying the sMap analysis (without any information on the states of the characters).
- `Mean posteriors` produces a plot of the mean posterior probabilities for each node. This is obtained for each node by considering all the trees that have the node, and computing the average of the posterior probabilities for each tree (i.e. without taking into account whether the node existed at a particular point in time or not).
- `Mean priors` does the same thing as Mean posteriors, but for prior probabilities.
- `Mean cond. posteriors` plots the mean posterior probabilities for each node, conditioned on the node existing at the time where it's found on the summary tree (this essentially the same as a Stochastic map plot, but without the branch states).
- `Stochastic map` is the default, and produces a stochastic map plot with posterior probabilities for the states of each node and each branch in the tree.
- `Sample sizes` produces a plot of the sample size for each point of each branch (i.e. for each branch, how many of the trees that have been used for the analysis actually have that branch at each point in time).
- `Active characters` determines the states of which characters are plotted. Especially in cases with many character (or many states), the plot can get confused: in these cases it is often better to produce separate plots for the various characters. To determine which characters are enabled, navigate with the arrows to reach each character, and press enter

(or the spacebar) to toggle its state (√ is active, □ is inactive). At least one character needs to be enabled at all times.

To close this dialog, highlight the desired plot target and press enter to save changes, or press esc to discard the changes.

6.7.1.2 Plot width

This field determines the width (in points) of the plot.

6.7.1.3 Plot height

This field determines the height (in points) of the plot. When the program is first opened, it is initialised to a value that should comfortably accommodate all of the tree labels.

6.7.1.4 Margins

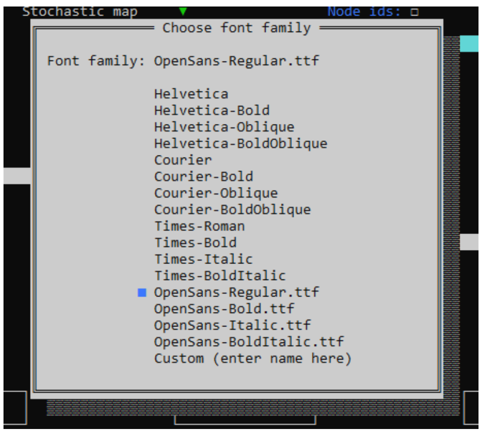
This field determines the whitespace margin (in points) on all sides of the plot.

6.7.1.5 Font size

This field determined the size (in points) of the font used to display the legend and the tree labels.

6.7.1.6 Font family

This field determines which typeface is used to for the tree labels and the legend. When the value of this field is edited, a dialog with all the possible fonts opens (shown on the right). The chosen font can be highlighted with the arrow keys, and enter confirms the choice, while esc cancels all changes.



```
Stochastic map                  Node ids: □
      ═══ Choose font family ═══

 Font family: OpenSans-Regular.ttf

              Helvetica
              Helvetica-Bold
              Helvetica-Oblique
              Helvetica-BoldOblique
              Courier
              Courier-Bold
              Courier-Oblique
              Courier-BoldOblique
              Times-Roman
              Times-Bold
              Times-Italic
              Times-BoldItalic
            ■ OpenSans-Regular.ttf
              OpenSans-Bold.ttf
              OpenSans-Italic.ttf
              OpenSans-BoldItalic.ttf
              Custom (enter name here)
```

To use a custom TrueType font, the `Custom font` option should be selected, and then the path to the font `.ttf` file should be provided (which can be a full path, or relative to the Plot-sMap executable, or relative to the current working directory).

6.7.1.7 Pie size

This field determines the radius (in points) of the pie charts used to display node probabilities in the plot (if the selected `Plot target` does not display pie charts, this setting has no effect).

6.7.1.8 Branch width

This field determines the size (in points) of the branch plots used to display the probabilities (or sample sizes) along branches (if the selected `Plot target` does not display branch plots, this setting has no effect).

6.7.1.9 Line width

This field determines the thickness (in points) of the lines used to draw the phylogenetic tree and the scale axis (if enabled).

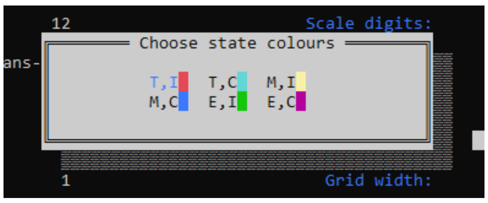
6.7.1.10 Time resolution

This field determines the resolution (in points) of the branch plots (if the selected `Plot target` does not display branch plots, this setting has no effect). Excessively small values produce very large plot files and take longer to execute.

6.7.1.11 Node ids

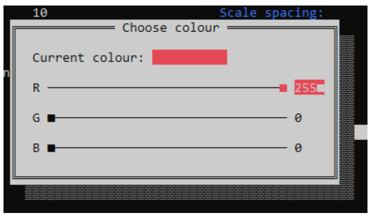
This field determines whether node ids are displayed above each node in the tree.

6.7.1.12 State colour



```
12                                   Scale digits:
ans-  ═══ Choose state colours ═══

        T,I🟥  T,C🟦  M,I🟨
        M,C🟦  E,I🟩  E,C🟪

  1                                  Grid width:
```

This field determines the colours associated to each state of the enabled character(s). When the value of this field is edited, a dialog is opened (shown to the right), displaying the available states and the colour associated to each (since the colour depth of the console is very low, these colours are only approximations; furthermore, if an unusual colour theme is enabled, they may be completely different from the real colours).



```
10                               Scale spacing:
      ═══ Choose colour ═══

  Current colour: 🟥

  R ─────────────────────────────■ 255
  G ■───────────────────────────── 0
  B ■───────────────────────────── 0
```

To change a colour for a state, the state should be highlighted using the arrow keys; pressing the spacebar then opens the colour choice dialog (shown to the right). This dialog can be used to supply the RGB values for the colour (in a scale from 0 to 255). The colour component can be chosen with the up and down arrows, while the value can be changed by the left and right arrows, or by using the keyboard to delete the current value and enter a new one. The enter key closes this dialog.

To save the changes to the state colours, the enter key can be used; the esc key, instead, discards all changes.

6.7.1.13 Scale axis

This field enables the time scale axis in the plot.

6.7.1.14 Scale spacing

This field determines the spacing (in units of time) between ticks on the time scale axis (if enabled). It is initialised by default to a value that should result in 5 ticks being displayed.

6.7.1.15 Scale digits

This field determines how many significant digits should be shown for the ages of the time scale axis ticks (if enabled).

6.7.1.16 Scale grid

This field enables the display of a time scale grid of vertical lines under the plot.

6.7.1.17 Grid spacing

This field determines the spacing (in units of time) between the lines of the time scale grid. It is initialised by default to a value that should result in 10 lines being drawn.

6.7.1.18 Grid colour

This field determines the colour of the lines of the time scale grid. When the value of this field is edited, a colour choice dialog similar to the one shown in 6.7.1.12 is displayed; the new value of the colour can be chosen as explained therein, and enter can be used to confirm (or esc to cancel).

6.7.1.19 Grid width

This field determines the thickness (in points) of the lines of the time scale grid.

6.7.1.20 Tree age scale

This field provides a number which used to re-normalise all ages on the time scale. It is initialised by default to the same value that was used to normalise the trees in the sMap analysis (if the trees were normalised, otherwise it is equal to 1; see the `-N`/`--norm` parameter of the sMap program in 6.1.5).

6.7.1.21 Plot…

This button can be used to save the plot as a PDF file. Upon highlighting it and pressing enter, a dialog is shown asking for the name of the output file (which is determined by default by looking at the input file name). This name can be changed using the keyboard; once an appropriate name has been chosen, the enter key is used to confirm and produce the plot (while the esc key cancels the operation).

Note that the program does not exit after saving the plot, which makes it possible to make further alterations to the values of the various fields and see what effect they have on the plot.

6.7.1.22 Save settings…

This button can be used to save the currently selected settings for the plot (these settings can be then recovered by providing the output file to the `-b/--batch` parameter, see 6.7.2). Similarly to the `Plot…` button, when this button is activated a dialog asks for the name of the output file.

6.7.1.23 Exit

This button is used to close the program. Upon highlighting it and pressing enter, Plot-sMap exits.

6.7.2 Command-line parameters

Plot-sMap has many command-line parameters, that can be used to specify all aspects of the plot (similarly to what can be done with the interactive interface). This can be useful when it's necessary to produce a plot without opening the program in interactive mode.

Parameter	Description
`-h` **`--help`**	**Optional**. Shows the help message, with a description of the command line parameters, and exits.

`-s <file>`
`--smap <file>`
	Required. Specifies the input sMap file. This should be a `sMap.bin` file produced by the sMap program.

`-b [settingsfile]`
`--batch [settingsfile]`
	Optional. Draw the plot with default settings and exit. If an optional settings file is provided (such as the one produced with the Save settings button, see 6.7.1.22), the settings from this file are used.

`-i` **`--interactive`**	**Optional**. When specified along with the `-b/--batch` parameter, this forces the program to run in interactive mode, while still loading the settings from the settings file provided to the `-b/--batch` parameter.

`-o <outputfile>`
`--output <outputfile>`
	Optional. Provides the output file name. **Default**: determined by the input file name.

`-t <plot>`
`--target <plot>`
	Optional. Determined the plot target (see 6.7.1.1). Available values are `Tree`, `MeanPriors`, `MeanPosteriors`, `MeanCondPosteriors`, `StochasticMap`, `SampleSizes`. **Default**: `StochasticMap`.

`-w <plot-width>`
`--width <plot-width>`
	Optional. Plot width (see 6.7.1.2). **Default**: 500.

`-h <plot-height>`
`--height <plot-height>`
	Optional. Plot width (see 6.7.1.3). **Default**: determined by the number of taxa.

`-m <margin>`
`--margins <margin>`
	Optional. Page margins (see 6.7.1.4). **Default**: 10.

`-f <font-size>`
`--font-size <font-size>`
	Optional. Font size (see 6.7.1.5). **Default**: 12.

`-F <font-family>`
`--font-family <font-family>`
	Optional. Font family (see 6.7.1.6). Available standard font families are: `Helvetica`, `Helvetica-Bold`, `Helvetica-Oblique`, `Helvetica-BoldOblique`, `Courier`, `Courier-Bold`, `Courier-Oblique`, `Courier-BoldOblique`, `Times-Roman`, `Times-Bold`, `Times-Italic`, `Times-BoldItalic`, `OpenSans-`

	`Regular.ttf`, `OpenSans-Bold.ttf`, `OpenSans-Italic.ttf`, `OpenSans-BoldItalic.ttf`. **Default**: `OpenSans-Regular.ttf`.

`-p <size>`
`--pie-size <size>`
	Optional. Pie chart radius (see 6.7.1.7). **Default**: 5.

`-B <width>`
`--branch-width <width>`
	Optional. Branch plot size (see 6.7.1.8). **Default**: 3.

`-l <width>`
`--line-width <width>`
	Optional. Plot line width (see 6.7.1.9). **Default**: 1.

`-n` `--node-ids`	**Optional**. Show node ids in the plot (see 6.7.1.11). **Default**: disabled.
`-x` `--scale-axis`	**Optional**. Show scale axis at the bottom of the plot (see 6.7.1.13). **Default**: disabled.

`-S <spacing>`
`--scale-spacing <spacing>`
	Optional. Scale axis spacing (see 6.7.1.14). **Default**: $\frac{\text{mean tree height}}{5}$.

`-g` `--scale-grid`	**Optional**. Show scale grid behind the plot (see 6.7.1.16). **Default**: disabled.

`-G <spacing>`
`--grid-spacing <spacing>`
	Optional. Scale grid spacing (see 6.7.1.17). Default: $\frac{\text{mean tree height}}{10}$.

`--gw <width>`
`--grid-line-width <width>`
	Optional. Scale grid line width (see 6.7.1.17). Default: 0.5.

`--gc <r>,<g>,<b>`
`--grid-colour <r>,<g>,<b>`
	Optional. Scale grid line colour, in RGB format (see 6.7.1.18). **Default**: 200,200,200.

`-d <digits>`
`--digits <digits>`
	Optional. Significant digits on the scale axis (see 6.7.1.15). **Default**: 3.

`-a <scale>`
`--age-scale <scale>`
	Optional. Tree age scale multiplier (see 6.7.1.20). **Default**: same value used to normalise trees in sMap.

`-r <resolution>`
`--resolution <resolution>`
	Optional. Branch time resolution (see 6.7.1.10). **Default**: $\frac{\text{plot width}}{250}$.

`-c <chars>`
`--active-characters <chars>`
	Optional. Comma-separated list of active characters (e.g. `0,2`; see 6.7.1.1). **Default**: all.

`-C <colours>`
`--state-colours <colours>`
	Optional. Colon-separated list of state colours (see 6.7.1.12) in RGB format between square brackets (e.g. `[255,0,0]:[0,255,255]`). **Default**: auto (determined by the number of states).

7 Licensing information

7.1 sMap licence

sMap is licensed under the GNU General Public Licence version 3, available at https://www.gnu.org/licenses/gpl-3.0.en.html.

7.2 Libraries used by sMap

7.2.1 Math.NET Numerics

sMap uses the Math.NET Numerics library to implement probability distributions and linear algebra computations. This library is released under the MIT/X11 licence, available at https://numerics.mathdotnet.com/License.html.

7.2.2 Newtonsoft.Json

sMap uses the Newtonsoft.Json library to produce the sMap output files containing the results of sMap analyses. This library is released under the MIT licence, available at https://licenses.nuget.org/MIT.

7.2.3 Mono.Options

sMap uses the Mono.Options library to parse command-line parameters. This library is released under the MIT licence, available at https://github.com/xamarin/XamarinComponents/blob/master/XPlat/Mono.Options/License.md.

7.2.4 Matrix exponential

sMap uses code from GitHub user *horribleheffalump* to compute matrix exponential of non-diagonalizable matrices. This code is released under the MIT licence, available at https://github.com/horribleheffalump/MatrixExponential/blob/master/LICENSE.txt.

7.2.5 HSLColor

sMap uses code from https://richnewman.wordpress.com/about/code-listings-and-diagrams/hslcolor-class/ to convert colours between HSL and RGB formats. This code is released in the public domain (https://richnewman.wordpress.com/about/code-listings-and-diagrams/hslcolor-class/#comment-8107).

7.2.6 BEAST

sMap uses code derived from BEAST (https://github.com/beast-dev/beast-mcmc/blob/v1.10.3/src/dr/inference/trace/TraceCorrelation.java) to compute effective sample sizes (ESS). This code is released under the GNU Lesser General Public Licence version 2, available at https://www.gnu.org/licenses/old-licenses/lgpl-2.0-standalone.html.

7.2.7 Avalonia

The graphical user interface of sMap-GUI is built using Avalonia (http://avaloniaui.net/), which is released under the MIT licence, available at https://github.com/AvaloniaUI/Avalonia/blob/master/licence.md.

7.2.8 GhostScript

sMap ships with a compiled version of GhostScript v9.27 (https://www.ghostscript.com/), which is used by the GUI version of Plot-sMap to provide plot previews. This is released under the GNU Affero General Public Licence version 3 (available at https://www.gnu.org/licenses/agpl-3.0.html). The source code for GhostScript can be found at: https://github.com/ArtifexSoftware/ghostpdl-downloads/releases/download/gs927/ghostscript-9.27.tar.gz.

8 References

Akaike, H. (1998) 'Information Theory and an Extension of the Maximum Likelihood Principle', in. Springer, New York, NY, pp. 199–213. doi: 10.1007/978-1-4612-1694-0_15.

Bollback, J. (2006) 'SIMMAP: Stochastic character mapping of discrete traits on phylogenies', *BMC Bioinformatics*. BioMed Central, 7(1), p. 88. doi: 10.1186/1471-2105-7-88.

Burnham, K. P. and Anderson, D. R. (2004) 'Multimodel Inference', *Sociological Methods & Research*. Sage PublicationsSage CA: Thousand Oaks, CA, 33(2), pp. 261–304. doi: 10.1177/0049124104268644.

Camacho, C. *et al.* (2009) 'BLAST+: architecture and applications', *BMC Bioinformatics*, 10(1), p. 421. doi: 10.1186/1471-2105-10-421.

Felsenstein, J. (1973) 'Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters', *Systematic Biology*. Narnia, 22(3), pp. 240–249. doi: 10.1093/sysbio/22.3.240.

Felsenstein, J. (1981) 'Evolutionary trees from DNA sequences: A maximum likelihood approach', *Journal of Molecular Evolution*. Springer-Verlag, 17(6), pp. 368–376. doi: 10.1007/BF01734359.

Felsenstein, J. (1986) *The Newick tree format*. Available at:

http://evolution.genetics.washington.edu/phylip/newicktree.html (Accessed: 19 February 2019).

Gelman, A. *et al.* (2013) *Bayesian Data Analysis*. CRC Press. Available at: https://www.crcpress.com/Bayesian-Data-Analysis/Gelman-Carlin-Stern-Dunson-Vehtari-Rubin/p/book/9781439840955.

Graham, S. W. *et al.* (1998) 'Phylogenetic Congruence and Discordance Among One Morphological and Three Molecular Data Sets from Pontederiaceae', *Systematic Biology*. Edited by D. Baum, 47(4), pp. 545–567. doi: 10.1080/106351598260572.

Huelsenbeck, J. P., Nielsen, R. and Bollback, J. P. (2003) 'Stochastic Mapping of Morphological Characters', *Systematic Biology*. Edited by T. Schultz. Oxford University Press, 52(2), pp. 131–158. doi: 10.1080/10635150390192780.

Katoh, K. and Standley, D. M. (2013) 'MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability', *Molecular Biology and Evolution*. Oxford University Press, 30(4), pp. 772–780. doi: 10.1093/molbev/mst010.

Kohn, J. R. *et al.* (1996) 'Reconstruction of the evolution of reproductive characters in Pontederiaceae using phylogenetic evidence from chloroplast DNA restriction-site variation', *Evolution*. John Wiley & Sons, Ltd (10.1111), 50(4), pp. 1454–1469. doi: 10.1111/j.1558-5646.1996.tb03919.x.

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) 'Nexus: An Extensible File Format for Systematic Information', *Systematic Biology*. Edited by D. Cannatella. Narnia, 46(4), pp. 590–621. doi: 10.1093/sysbio/46.4.590.

Nesterov, Y. (2004) *Introductory Lectures on Convex Optimization*. Boston, MA: Springer US (Applied Optimization). doi: 10.1007/978-1-4419-8853-9.

Nielsen, R. (2002) 'Mapping Mutations on Phylogenies', *Systematic Biology*. Edited by J. Huelsenbeck. Oxford University Press, 51(5), pp. 729–739. doi: 10.1080/10635150290102393.

Raftery, A. E. (1995) 'Bayesian Model Selection in Social Research', *Sociological Methodology*. American Sociological Association, 25, p. 111. doi: 10.2307/271063.

Rambaut, A. *et al.* (2018) 'Posterior Summarization in Bayesian Phylogenetics Using Tracer 1.7', *Systematic Biology*. Edited by E. Susko. Narnia, 67(5), pp. 901–904. doi: 10.1093/sysbio/syy032.

Revell, L. J. (2012) 'phytools: an R package for phylogenetic comparative biology (and other things)', *Methods in Ecology and Evolution*. John Wiley & Sons, Ltd (10.1111), 3(2), pp. 217–223. doi: 10.1111/j.2041-210X.2011.00169.x.

Ronquist, F. and Huelsenbeck, J. P. (2003) 'MrBayes 3: Bayesian phylogenetic inference under mixed models', *Bioinformatics*. Oxford University Press, 19(12), pp. 1572–1574. doi: 10.1093/bioinformatics/btg180.

Schwarz, G. (1978) 'Estimating the Dimension of a Model', *The Annals of Statistics*. Institute of Mathematical Statistics, 6(2), pp. 461–464. doi: 10.1214/aos/1176344136.

Stern, D. L. (1998) 'Phylogeny of the tribe Cerataphidini (Homoptera) and the evolution of the horned soldier aphids', *Evolution*. John Wiley & Sons, Ltd (10.1111), 52(1), pp. 155–165. doi: 10.1111/j.1558-5646.1998.tb05148.x.

Tavaré, S. (1986) 'Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences', in *American Mathematical Society: Lectures on Mathematics in the Life Sciences*. Amer Mathematical Society, pp. 57–86. Available at: citeulike-article-id:4801403.

Xie, W. *et al.* (2011) 'Improving Marginal Likelihood Estimation for Bayesian Phylogenetic Model Selection', *Systematic Biology*. Narnia, 60(2), pp. 150–160. doi: 10.1093/sysbio/syq085.