Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling

Jan Kudlicka
Uppsala University
Uppsala, Sweden

Lawrence M. Murray
Uber AI Labs
San Francisco, CA, USA

Fredrik Ronquist
Swedish Museum
of Natural History
Stockholm, Sweden

Thomas B. Schön
Uppsala University
Uppsala, Sweden

Abstract

We consider probabilistic programming for birth-death models of evolution and introduce a new widely-applicable inference method that combines an extension of the alive particle filter (APF) with automatic Rao-Blackwellization via delayed sampling. Birth-death models of evolution are an important family of phylogenetic models of the diversification processes that lead to evolutionary trees. Probabilistic programming languages (PPLs) give phylogeneticists a new and exciting tool: their models can be implemented as probabilistic programs with just a basic knowledge of programming. The general inference methods in PPLs reduce the need for external experts, allow quick prototyping and testing, and accelerate the development and deployment of new models. We show how these birth-death models can be implemented as simple programs in existing PPLs, and demonstrate the usefulness of the proposed inference method for such models. For the popular BiSSE model the method yields an increase of the effective sample size and the conditional acceptance rate by a factor of 30 in comparison with a standard bootstrap particle filter. Although concentrating on phylogenetics, the extended APF is a general inference method that shows its strength in situations where particles are often assigned zero weight. In the case when the weights are always positive, the extra cost of using the APF rather than the bootstrap particle filter is negligible, making our method a suitable drop-in replacement for the bootstrap particle filter in probabilistic programming inference.

1 INTRODUCTION

The development of new probabilistic models of evolution is an important part of statistical phylogenetics. These models require inference algorithms that are able to cope with increased model complexity as well as the larger amount of observational data available today. Experts from several fields typically need to be involved, both to design bespoke inference algorithms, and to implement the new models and the inference algorithms in existing software or to develop new software from scratch. Probabilistic programming languages (PPLs) (e.g., Goodman et al., 2008; Tolpin et al., 2016; Mansinghka et al., 2014; Paige and Wood, 2014) have the potential to accelerate this: generative models are specified as simple programs and compiled into executable applications that include general inference engines. Writing models in PPLs requires just basic programming skills, and thus allows quick prototyping and testing.

Quite a few software applications for statistical phylogenetics exist today, including the popular MrBayes (Huelsenbeck and Ronquist, 2001) and BEAST (Drummond and Rambaut, 2007). They typically take a Bayesian approach and implement Markov chain Monte Carlo inference (see review by Nascimento et al., 2017). Most of these applications do not allow the user to specify models outside of a predefined model space, which can be quite narrow. Even when adding new models is possible, it is usually a challenging task requiring not only good programming skills but also detailed knowledge of the software design and implementation.

Statistical phylogeneticists recognize the benefits of software that supports the addition of new models and inference methods. For example, the design of BEAST 2 (Bouckaert et al., 2014) allows users to create and use custom modules. RevBayes (Höhna et al., 2016) goes even further: it uses a domain-specific probabilistic programming language for phylogenetics based on probabilistic graphical models (e.g., Koller and Friedman,

2009). However, the language is not Turing-complete, which means it has some limitations. For example it does not allow unbounded recursion.

In this paper we concentrate on birth-death models of evolution, an important family of phylogenetic models. In these models, births correspond to lineage splits (speciation events) and deaths to extinction events. These models specify probability distributions of evolutionary trees and the task is to infer model parameters given a part of a complete tree that represents evolution of currently living species.

We take a step toward using PPLs in statistical phylogenetics: our main contribution is a new general inference algorithm based on an extension of the alive particle filter (APF) (Del Moral et al., 2015) combined with automatic Rao-Blackwellization via delayed sampling (Murray et al., 2018). We also show how to implement birth-death models of evolution in existing PPLs, and show the usefulness of our inference algorithm for such models. Interestingly, by using this algorithm we avoid sampling of birth and death rates. We believe that the algorithm may be of interest for other models with highly-informative observations. Finally, we prove that the estimator of the marginal likelihood in the extended APF is unbiased.

The rest of the paper is organized as follows: in Section 2 we give a brief recapitulation of basic concepts in evolution and introduce probabilistic programming in more detail. We derive our inference algorithm and show how phylogenetic birth-death algorithms can be implemented in PPLs in Section 3. We give implementations of two well-known phylogenetic birth-death models and compare several general inference algorithms for these models in Section 4. We offer some conclusions and ideas for future research in Section 5.

2 BACKGROUND

2.1 SPECIATION, EXTINCTION AND PHYLOGENIES

There are two types of events that play a significant role in the evolution of any species. *Speciation* occurs when the population of one species splits and eventually forms two new species. *Extinction* occurs when the whole population of one species dies out. Species that are not extinct, i.e., species with individuals alive at the present time, are called *extant*.

In phylogenetics, the *before present* (BP) time is usually used for dating, i.e., if an event happened at time τ it means it happened τ time units ago.

The result of an evolutionary process is a binary tree called the *complete phylogeny*. A very simple example of a complete phylogeny is depicted in Figure 1a. The nodes represent events and species at significant times:

- the root node represents the most recent common ancestor (MRCA) of all species of interest,
- an internal node represents a speciation event,
- a leaf at $\tau = 0$ (i.e. the present time) represents an extant species,
- a leaf at $\tau > 0$ (i.e. in the past) represents an extinction event.

The length of edges—or *branches* as they are called in phylogenetics—is the difference between the time of the parent and the child node.

The *reconstructed* phylogeny is obtained from a complete tree by removing all subtrees that involve only extinct species. We will refer to this as pruning. An example of a reconstructed tree is depicted in Figure 1b.

The reconstructed phylogeny represents the evolution of the extant species and only contains information that can be observed directly (the extant species) or *reconstructed* by statistical analysis of the DNA sequences of extant species (the topology of the tree and the times of the speciation events).

2.2 PROBABILISTIC PROGRAMMING

The development of new probabilistic models and inference algorithms is a time-consuming and possibly error-prone process that usually requires skilled experts in probability, statistics and computer science. Probabilistic programming is a relatively new approach to solve this problem: generative models are expressed as computer programs in probabilistic programming languages (PPLs) with support for random variables and operations on them. Integral to PPLs are general inference engines that perform the inference in such programs. These engines estimate the distribution of all latent random variables conditioned on the observed data and use it to answer the queries of interest.

PPLs allow us to define and initialize random variables with a given distribution, for example:

$$x \sim \mathrm{Normal}(0, 1)$$

The program may use random variables as though any ordinary variable, and control the flow of the execution. Depending on the PPL, conditioning on the observed data might be specified explicitly or implicitly. The former means that conditioning on the observed data is a part of the program, for example:

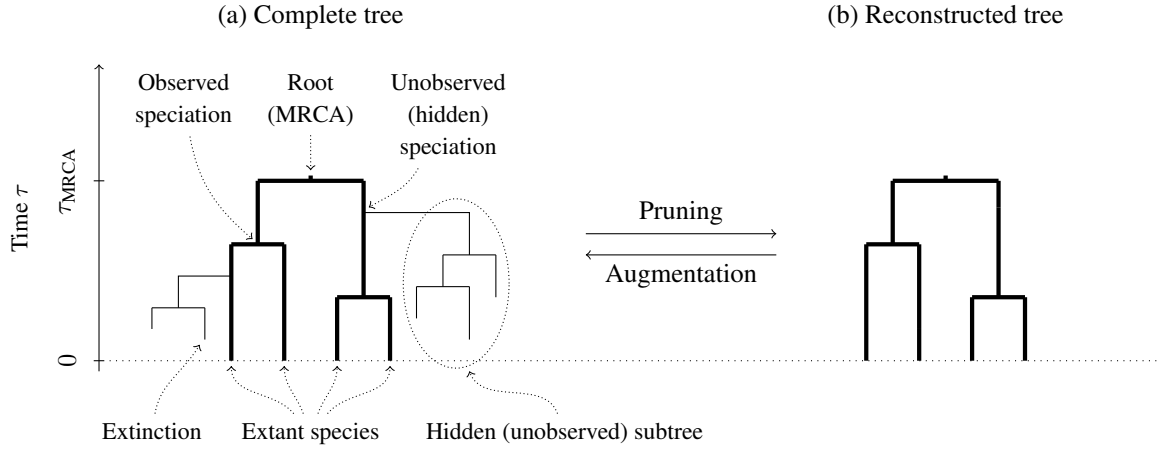


Figure 1: Complete (on the left) vs. reconstructed (on the right) tree. The reconstructed tree shows only the evolution of the extant species.

$x \sim \text{Normal}(0, 1)$
observe $0.892 \sim \text{Normal}(x, 1)$

The latter implies that observed values of random variables are not part of the program, but instead specified at run time (e.g. as arguments).

The main general inference methods used in PPLs are adaptations of various inference algorithms for the universal setting described below, including Markov chain Monte Carlo (MCMC) (Metropolis et al., 1953; Hastings, 1970), sequential Monte Carlo (SMC) (Gordon et al., 1993; Del Moral et al., 2006), and Hamiltonian Monte Carlo (HMC) (Neal, 2011).

There already exist quite a few PPLs today based on different programming paradigms, for example functional PPLs like Anglican (Tolpin et al., 2016) and Venture (Mansinghka et al., 2014); imperative Probabilistic C (Paige and Wood, 2014), Turing (Ge et al., 2018), Stan (Carpenter et al., 2017), Edward (Tran et al., 2016) and Pyro (Bingham et al., 2019); and object-oriented Birch (Murray and Schön, 2018).

2.3 PROGRAMMATIC MODEL AND SMC

The execution of a probabilistic program can be modeled using a *programmatic model* (Murray and Schön, 2018). Let $\{V_i\}_i$ denote a countable set of all random variables, both latent and observed, in a probabilistic program. This set might be infinite due to loops and recursion. During execution, whenever a random variable V_i is encountered, its realization v_i is drawn from a distribution associated with it.

Multiple executions of the program might in general encounter different subsets of the random variables (e.g., due to using random variables in conditional expressions) and encounter them in a different order (with the exception of the first one). For each random variable V_j not encountered during the execution we set $v_j = \perp$ (the symbol $\perp$ represents an *undefined* value). We will however assume that any execution encounters all *observed* random variables and that these observations are encountered in the same order.

Let σ denote a sequence of indices into $\{V_i\}$ specifying the order in which the random variables are encountered during an execution of the program, and let $|\sigma|$ denote the length of this sequence. Also, let $\{v_i\}_{i\in\sigma}$ denote the realizations of the random variables indexed by σ, i.e., $v_{\sigma[1]}, \ldots, v_{\sigma[|\sigma|]}$. In a similar manner, we will use $\{V_i = v_i\}_{i\in\sigma}$ to denote $V_{\sigma[1]} = v_{\sigma[1]}, \ldots, V_{\sigma[|\sigma|]} = v_{\sigma[|\sigma|]}$.

The index of the k-th encountered variable is given by a deterministic function Ne (for *next*) of the realizations of the previously encountered random variables, so that

$$\sigma[k] = \text{Ne}(\{v_i\}_{i\in\sigma[1:k-1]}),$$

where $\sigma[1{:}k{-}1]$ denotes the indices of the first $k-1$ encountered random variables. The function Ne is uniquely defined by the probabilistic program. If there are no more random variables to encounter, Ne returns $\perp$.

The k-th encountered random variable, $V_{\sigma[k]}$, is sampled from

$$V_{\sigma[k]} \sim p_{\sigma[k]}(\cdot \,|\, \text{Pa}(\{v_i\}_{i\in\sigma[1:k-1]})),$$

where $p_{\sigma[k]}$ is the distribution specified by the program, Pa (for *parents*) is a deterministic function returning the parameters of this distribution, and again, it is a function of the realizations of the previously encountered random variables.

Algorithm 1 Bootstrap particle filter (BPF).

for $n = 1$ **to** N **do** ▷ Initialize
  $v_0^{(n)} \leftarrow \varnothing$; $w_0^{(n)} \leftarrow 1/N$
for $t = 1$ **to** T **do**
  for $n = 1$ **to** N **do**
    $a \sim \mathrm{Cat}(\{w_{t-1}^{(m)} / \sum_{l=1}^{N} w_{t-1}^{(l)}\}_{m=1}^{N})$ ▷ Resample
    $v_t^{(n)} \leftarrow \textsc{Propagate}(v_{t-1}^{(a)})$ ▷ Propagate
    $w_t^{(n)} \leftarrow p_{\gamma[t]}(y_t | \mathrm{Pa}(v_t^{(n)}))$ ▷ Weigh
    $v_t^{(n)}[\gamma[t]] \leftarrow y_t$

function $\textsc{Propagate}(v)$ ▷ Run until next observe
  $\kappa \leftarrow \mathrm{Ne}(v)$
  while $\kappa \notin \gamma$ **do**
    $v[\kappa] \sim p_\kappa(\cdot | \mathrm{Pa}(v))$
    $\kappa \leftarrow \mathrm{Ne}(v)$
  return v

The joint distribution function encoded by the program can be given recursively (starting with $\sigma = []$):

$$p(\{v_i\}_{i \notin \sigma} | \sigma, \{V_j = v_j\}_{j \in \sigma}) = \begin{cases} p(\{v_i\}_{i \notin \sigma'} | \sigma', \{V_j = v_j\}_{j \in \sigma'}) \\ \quad \times p_\kappa(v_\kappa | \mathrm{Pa}(\{v_j\}_{j \in \sigma})) & \text{if } \kappa \neq \perp, \\ 1 & \text{if } \kappa = \perp \wedge \forall i \notin \sigma : v_i = \perp, \\ 0 & \text{otherwise,} \end{cases}$$

where $\kappa = \mathrm{Ne}(\{v_i\}_{i \in \sigma})$ and σ' is obtained from σ by appending κ. The first case is the conditional probability chain rule, the remaining cases cover the situation where there are no more random variables to encounter.

We wish to sample from the posterior distribution $p(\{v_i\}_{i \notin \gamma} | V_{\gamma[1]} = y_1, \ldots, V_{\gamma[T]} = y_T)$, where T denotes the number of observations, y_t denotes the t-th observation and γ denotes the sequence of indices of the observed random variables in $\{V_i\}$. The sequential nature of the joint distribution allows us to employ Sequential Monte Carlo methods (Del Moral et al., 2006) to sample from this posterior distribution, including the *bootstrap particle filter* (BPF) summarized in Algorithm 1. For the sake of brevity we have assumed that the last observation is also the last encountered random variable. In the pseudocode, $\mathrm{Cat}()$ denotes the categorical distribution with the given event probabilities. Variables denoted by v are associative arrays (also known as maps or dictionaries) used to store the realizations of random variables ($v[i]$ denotes the realization of V_i). The PROPAGATE function runs the program until it encounters an observation.

Samples from the joint distribution and the corresponding weights can be used to estimate the expected value of a test function h of interest:

$$\widehat{\mathbb{E}}[h] = \frac{\sum_n w_T^{(n)} h\left(v_T^{(n)}\right)}{\sum_n w_T^{(n)}},$$

as well as to estimate the marginal likelihood $p(y_{1:T})$:

$$\widehat{Z} = \prod_{t=1}^{T} \frac{1}{N} \sum_{n=1}^{N} w_t^{(n)}.$$

3 METHODS

3.1 EXTENDED ALIVE PARTICLE FILTER

In the bootstrap particle filter, each particle is propagated by simulating the prior, and may make random choices that lead to a state with zero weight. In phylogenetic birth-death models this happens quite often: when simulating the evolution of subtrees that must ultimately become extinct, if any species happen to survive to the present time, the particle is assigned zero weight. In extreme cases, all particles have zero weight, and the BPF degenerates.

Del Moral et al. (2015) considered this problem in a setting with indicator potentials (such as in approximate Bayesian computation), i.e. all weights being either zero or one. They proposed a modification of the BPF, where the resampling and propagation steps are repeated for particles that have weight zero until all particles have weight one. Details of the resulting alive particle filter (APF) as well as proofs of some of its theoretical properties can be found in Del Moral et al. (2015).

Although the original APF was designed specifically for indicator potentials, we have *extended the algorithm to work with importance weights*, see Algorithm 2 (the PROPAGATE function is the same as in Algorithm 1). The APF requires $N + 1$ particles rather than N in order to estimate the marginal likelihood without bias.

At the t-th observe statement, if the weight of a particle is zero, the resampling and propagation steps are repeated. This procedure is repeated until the weights of all $N + 1$ particles are positive. The APF counts the total number of propagations P_t for each observation. The algorithm never uses the states or weights of the $N + 1$-th particle, but propagations made using this particle are included in P_t, and used to calculate the unbiased estimate of the marginal likelihood $p(y_{1:T})$:

$$\widehat{Z} = \prod_{t=1}^{T} \frac{\sum_{n=1}^{N} w_t^{(n)}}{P_t - 1}.$$

The proof of unbiasedness can be found in Appendix A in the supplementary material.

Algorithm 2 Alive particle filter (APF).

for $n = 1$ **to** N **do** ▷ Initialize
  $v_0^{(n)} \leftarrow \varnothing$; $w_0^{(n)} \leftarrow 1/N$
for $t = 1$ **to** T **do**
  $P_t \leftarrow 0$
  for $n = 1$ **to** $N + 1$ **do**
    repeat ▷ Resample
      $a \sim \mathrm{Cat}(\{w_{t-1}^{(m)} / \sum_{l=1}^{N} w_{t-1}^{(l)}\}_{m=1}^{N})$
      $v_t^{(n)} \leftarrow \textsc{Propagate}(v_{t-1}^{(a)})$ ▷ Propagate
      $P_t \leftarrow P_t + 1$
      $w_t^{(n)} \leftarrow p_{\gamma[t]}(y_t | \mathrm{Pa}(v_t^{(n)}))$ ▷ Weigh
    until $w_t^{(n)} > 0$
    $v_t^{(n)}[\gamma[t]] \leftarrow y_t$

Unbiasedness of the marginal likelihood estimate opens for the possibility to use the APF within particle Markov chain Monte Carlo methods.

3.2 BIRTH-DEATH MODELS AS PROBABILISTIC PROGRAMS

Phylogentic birth-death models constitute a family of models where speciation (birth) events and extinction (death) events occur along the branches of a phylogenetic tree. Typically, the waiting times between events are exponentially distributed. In general, the rates of these exponential distributions do not remain constant but rather change continuously, discontinuously, or both. Some models assume that these rates further depend on a state variable that itself evolves discontinuously along the tree; in some cases the value of this state variable is given for the extant species.

The *constant-rate birth death (CRBD)* model (Kendall, 1948) is the simplest birth-death model, where the speciation rate λ and extinction rate μ remain constant over time. Pseudocode for generating phylogenetic trees using the CRBD model can be found in Appendix B in the supplementary material.

Let $\mathcal{T}_r$ denote the subtree rooted at the node r, and $\mathrm{Ch}(r)$ denote the children of this node. The likelihood of the subtree $\mathcal{T}_r$ can be expressed recursively (we have dropped conditioning on λ and μ in the notation for brevity):

$$p(\mathcal{T}_r) = \begin{cases} 2 \prod_{c \in \mathrm{Ch}(r)} p(\mathcal{T}_c) & \text{if } r \text{ is the root node,} \\ 2\lambda e^{-(\lambda+\mu)\Delta_r} \prod_{c \in \mathrm{Ch}(r)} p(\mathcal{T}_c) & \text{if } r \text{ is a speciation,} \\ \mu e^{-(\lambda+\mu)\Delta_r} & \text{if } r \text{ is an extinction,} \\ e^{-(\lambda+\mu)\Delta_r} & \text{if } r \text{ is an extant species,} \end{cases}$$

where Δ_r is the length of the branch between the node r and its parent. The factor 2 occurs in the formulas because the orientation of the two child subtrees does not matter. If r is a speciation event, no extinction occurs along the branch (factor $e^{-\mu\Delta_r}$) and the speciation happens after a waiting time Δ_r (factor $\lambda \exp^{-\lambda\Delta_r}$). If r is an extinction event, no speciation occurs along the branch (factor $e^{-\lambda\Delta_r}$) and the extinction occurs after waiting time Δ_r (factor $\mu e^{-\mu\Delta_r}$). Finally, if r is an extant species, neither extinction nor speciation occurs along the branch. The likelihood of the complete phylogeny $\mathcal{T}$ is given by $p(\mathcal{T}_{\text{root}})$.

The likelihood in other birth-death models that admit varying rates and/or include the state can be derived in a similar way.

The likelihood of the complete phylogeny for the CRBD model can be conveniently written as

$$p(\mathcal{T}) = 2^{S+1} \lambda^S \mu^X e^{-(\lambda+\mu)L},$$

where L is the sum of all branch lengths, S the number of speciation events (excluding the root) and X the number of extinction events.

The task of interest is to infer model parameters given a reconstructed tree. Recall that this tree is a part of the complete tree corresponding to the extant species and their ancestors. A naive approach to inference is to simulate trees from the generative model, prune back the extinct subtrees, and reject those for which the pruned tree does not equal the observed tree. Such an approach always results in rejection.

Instead, we turn the problem upside down: starting with the observed tree and *augmenting* it with unobserved information to obtain a complete tree. Recalling Figure 1, the observed tree is traversed in depth-first order. At the root node the importance weight is doubled (since the orientation of the children does not matter). Along each branch, the generative model is used to simulate:

- changes to the state (in models with state),
- changes to the speciation and extinction rates,
- hidden speciation events.

For each of the hidden speciation events, the model simulates the evolution of the new species (i.e. a hidden subtree). If any portion of a hidden subtree survives to the present time, the weight is set to zero. If not, the current weight is doubled, since the children created by a hidden speciation event on an observed branch are interchangeable: either can correspond to the observed subtree, the other must correspond to the hidden subtree and goes extinct.

If the examined branch ends with a speciation event, the algorithm observes $0 \sim \text{Exponential}(\lambda)$ and doubles the weight. Finally, as there were no extinction events along the processed branch, the algorithm observes $0 \sim \text{Poisson}(\mu\Delta)$. In models with state, if the branch ends at $\tau = 0$ (i.e. the present time) we also condition on the simulated state being equal to the observed state. We trigger resampling at the end of each branch.

Let us return to the CRBD model in light of the discussion above. The likelihood of a proposed complete tree $\mathcal{T}'$ is given by

$$q(\mathcal{T}') = \lambda^{H'} e^{-\lambda L_{\text{obs}}} \times (2\lambda)^{S'} \mu^{X'} e^{-(\lambda+\mu)L'},$$

where H' denotes the number of all simulated hidden speciation events along the observed tree, L_{obs} the sum of the branch lengths in the observed tree, S' the number of speciation events in all hidden subtrees, X' the number of extinction events and finally L' denotes the sum of the branch lengths in the hidden subtrees. The factor $\lambda^{H'} e^{-\lambda L_{\text{obs}}}$ is related to the hidden speciation events, and the rest is the combined likelihood of all hidden subtrees.

The weight of the proposal $\mathcal{T}'$ that is compatible with the observation (i.e. without any extant species in the hidden subtrees) is given by

$$w(\mathcal{T}') = 2^{H'} \times 2^{S_{\text{obs}}+1} \times \lambda^{S_{\text{obs}}} \times e^{-\mu L_{\text{obs}}},$$

where S_{obs} is the number of observed speciation events. The factor $2^{H'}$ corresponds to doubling the weight for each hidden subtree, the factor $2^{S_{\text{obs}}+1}$ corresponds to doubling the weight at the root node and at all observed speciations, the factor $\lambda^{S_{\text{obs}}}$ is due to observing $0 \sim \text{Exponential}(\lambda)$ at all observed speciations, and finally the factor $e^{-\mu L_{\text{obs}}}$ is due to observing $0 \sim \text{Poisson}(\mu\Delta)$ for all observed branches.

Multiplying the likelihood $q(\mathcal{T}')$ and the weight $w(\mathcal{T}')$ of the proposal and summing the event numbers and the branch lengths we get

$$q(\mathcal{T}')w(\mathcal{T}') = p(\mathcal{T}').$$

3.3 DELAYED SAMPLING OF THE RATES

In a Bayesian setting, the parameters are associated with a prior distribution. Using the gamma distribution (or the exponential distribution as its special case) as a prior for the rates of speciation, extinction and state change is mathematically convenient since the gamma distribution is a conjugate prior for both the Poisson and the exponential likelihood. Instead of sampling these parameters from the prior distribution before running the particle filter (which we refer to as *immediate sampling*), we can exploit the conjugacy, which allows us to marginalize out the parameters and sample them after running the particle filter. Exploiting the conjugacy in a probabilistic program can be automated by an algorithm known as *delayed sampling* (Murray et al., 2018).

Consider the following prior:

$$\nu \sim \text{Gamma}(k, \theta),$$

with $k \in \mathbb{N}$. When the program needs to make a draw from a Poisson distribution

$$n \sim \text{Poisson}(\nu\Delta),$$

it can instead make a draw from the marginalized distribution:

$$n \sim \text{NegativeBinomial}\left(k, \frac{1}{1+\Delta\theta}\right),$$

where $\text{NegativeBinomial}(k, p)$ is the negative binomial distribution counting the number of failures given the number of successes k and the probability of success p in each trial. The distribution for ν is then updated to the posterior distribution according to

$$\nu \sim \text{Gamma}\left(k+n, \frac{\theta}{1+\Delta\theta}\right).$$

Similarly for variables distributed according to the exponential distribution, instead of drawing

$$\Delta \sim \text{Exponential}(\nu),$$

the program makes a draw from the marginalized distribution:

$$\Delta \sim \text{Lomax}\left(\frac{1}{\theta}, k\right),$$

where the first parameter denotes the scale and the second the shape of the Lomax distribution, and the distribution for ν is then updated to

$$\nu \sim \text{Gamma}\left(k+1, \frac{\theta}{1+\Delta\theta}\right).$$

Using this strategy there is actually *no need to sample the rates at all*; all draws involving these rates are replaced by draws from the negative binomial and the Lomax distributions with a consequent update of the rate distribution. Details of these conjugacy relationships can be found in Appendix C in the supplementary material.

4 EXPERIMENTS

We implemented the inference algorithms described above in the probabilistic programming language

Birch (Murray and Schön, 2018) and added support for the conjugacy relationships described in the previous section, so that Birch can provide automated delayed sampling for these. We also implemented two phylogenetic birth-death models, described in Sections 4.1 and 4.2 below.

We ran the inference for these models using different combinations of the inference method, the sampling strategy (immediate or delayed) and different number of particles N. For each combination we executed the program M times, collected the estimates $\{\widehat{Z}_m\}_{m=1}^M$ of the marginal likelihood and calculated the relative effective sample size (RESS):

$$\text{RESS} = \frac{1}{M} \frac{\left(\sum_{m=1}^M \widehat{Z}_m\right)^2}{\sum_{m=1}^M \widehat{Z}_m^2},$$

as well as the conditional acceptance ratio (CAR) (see Murray et al., 2013 for more detail):

$$\text{CAR} = \frac{1}{M}\left(2\sum_{i=1}^{M} c_i - 1\right),$$

where c_i is the sum of the i smallest elements in $\{\widehat{Z}_m\}_m$. We also calculated the sample variance $\operatorname{var} \log \widehat{Z}$.

For the experiments with the APF we also compared the total number of propagations with the number of propagations in the BPF by calculating

$$\rho = \frac{\sum_{m=1}^M P_m}{MNT},$$

where P_m is the number of all propagations made during the m-th execution and T is the number of branches in the observation. Note that NT is the number of propagations in the BPF.

4.1 CONSTANT-RATE BIRTH DEATH MODEL

Pseudocode for the probabilistic program implementing the *constant-rate birth death (CRBD)* model is listed as Algorithm 3. To sample speciation events along a branch we first sample a number of events from a Poisson distribution and then sample the time of each event from a uniform distribution. The implementation in Birch can be found in the supplementary material.

We used the phylogeny of cetaceans (Steeman et al., 2009) as the observation. This phylogeny (Figure 2 in the supplementary material) represents the evolution of whales, dolphins and porpoises and contains 87 extant species. We used $\text{Gamma}(1, 1)$ as the prior for both the speciation and extinction rate. The results of experiments comparing BPF and APF with immediate or delayed sampling for different number of particles N, and running $M = 200$ executions for each combination, are summarized in Table 1 and Figure 3a.

Algorithm 3 CRBD model as a probabilistic program.

Input:

- $\mathcal{T}$ – a pre-ordered list of nodes in the observation
- $k_\lambda, \theta_\lambda$ – the shape and scale of the prior Gamma distribution for λ ($k_\lambda \in \mathbb{N}$)
- k_μ, θ_μ – the shape and scale of the prior Gamma distribution for μ ($k_\mu \in \mathbb{N}$)

$\lambda \sim \text{Gamma}(k_\lambda, \theta_\lambda)$
$\mu \sim \text{Gamma}(k_\mu, \theta_\mu)$
for all $r \in \mathcal{T}$ **do**
  if r is the root **then**
    double the weight
    continue
  $c_{\text{hs}} \sim \text{Poisson}(\lambda\Delta_r)$
  for $i \leftarrow 1$ **to** c_{hs} **do**
    $\tau \sim \text{Uniform}(\tau_r, \tau_r + \Delta_r)$
    if BRANCHSURVIVES(τ) **then**
      set the weight to 0 and **return**
    else
      double the weight
  if r has children **then**
    observe $0 \sim \text{Exponential}(\lambda)$
    double the weight
  observe $0 \sim \text{Poisson}(\mu\Delta_r)$

function BRANCHSURVIVES(τ, λ, μ)
  $\Delta \sim \text{Exponential}(\mu)$
  if $\Delta \geq \tau$ **then**
    return true
  $c_b \sim \text{Poisson}\,(\lambda\Delta)$
  for $i \leftarrow 1$ **to** c_b **do**
    $\tau' \sim \text{Uniform}(\tau - \Delta, \tau)$
    if BRANCHSURVIVES(τ', λ, μ) **then**
      return true
  return false

When using delayed sampling, the speciation and extinction rates are never sampled; the rates are instead represented by gamma distributions with parameters that are updated during the execution. Let $k_\lambda^{(m)}, \theta_\lambda^{(m)}, k_\mu^{(m)}$ and $\theta_\mu^{(m)}$ denote the final values of these parameters for a particle drawn from all particles in the m-th run with the probabilities proportional to their final weights. The posterior distributions for λ and μ can be estimated by mixtures of gamma distributions:

$$\lambda \sim \frac{1}{\sum_{m=1}^M \widehat{Z}_m} \sum_{m=1}^{M} \widehat{Z}_m \,\text{Gamma}\left(k_\lambda^{(m)}, \theta_\lambda^{(m)}\right),$$

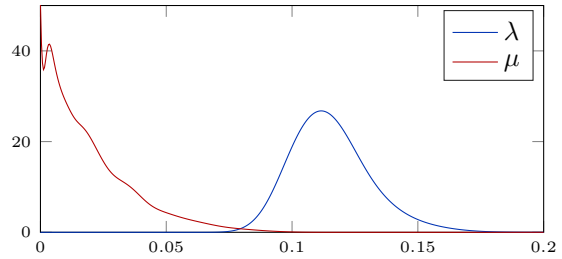


Figure 2: The posterior distributions for the speciation and extinction rates for the cetaceans using the CRBD model.

$$\mu \sim \frac{1}{\sum_{m=1}^{M} \widehat{Z}_m} \sum_{m=1}^{M} \widehat{Z}_m \operatorname{Gamma}\left(k_\mu^{(m)}, \theta_\mu^{(m)}\right).$$

Figure 2 depicts the posterior distributions for the speciation and extinction rates estimated using $M = 1000$ runs of the APF with $N = 4096$ particles.

4.2 BINARY-STATE SPECIATION AND EXTINCTION MODEL

The *binary-state speciation and extinction (BiSSE)* model (Maddison et al., 2007) introduces a binary state for species, denoted by $s \in \{0, 1\}$. Each state has its own (but constant) speciation and extinction rates, denoted by λ_s and μ_s. The waiting time between switching state is exponentially distributed with rates q_{01} for switching from state 0 to state 1, and q_{10} from state 1 to state 0. In our experiments we made a (common) assumption that $q_{01} = q_{10} = \varsigma$.

We used the same observation as Rabosky and Goldberg (2015), i.e., we extended the cetacean phylogeny with the state variable related to the body length of cetaceans obtained from Slater et al. (2010). Data are available for 74 of the 87 extant species. The binary state 0 and 1 refers to the length being below and above the median. Again we used $\operatorname{Gamma}(1, 1)$ as the prior for λ_0, λ_1, μ_0 and μ_1, and $\operatorname{Gamma}(1, 10/820.28)$ as the prior for ς (the number in the denominator is the sum of all branch lengths). The initial state at the root is drawn from $\{0, 1\}$ with equal probabilities. The results for experiments comparing the BPF and the APF with immediate or delayed sampling for different number of particles N, and running $M = 200$ executions for each combination, are summarized in Table 2 and Figure 3b. When running the experiments using the BPF and immediate sampling, a certain fraction of the executions degenerated—from 25% of the executions with 1024 particles down to 1.5% of the executions with 4096 particles. These executions were excluded when calculating $\operatorname{var} \log \widehat{Z}$.

Our implementation of the BiSSE model can be found in the supplementary material.

5 DISCUSSION AND CONCLUSION

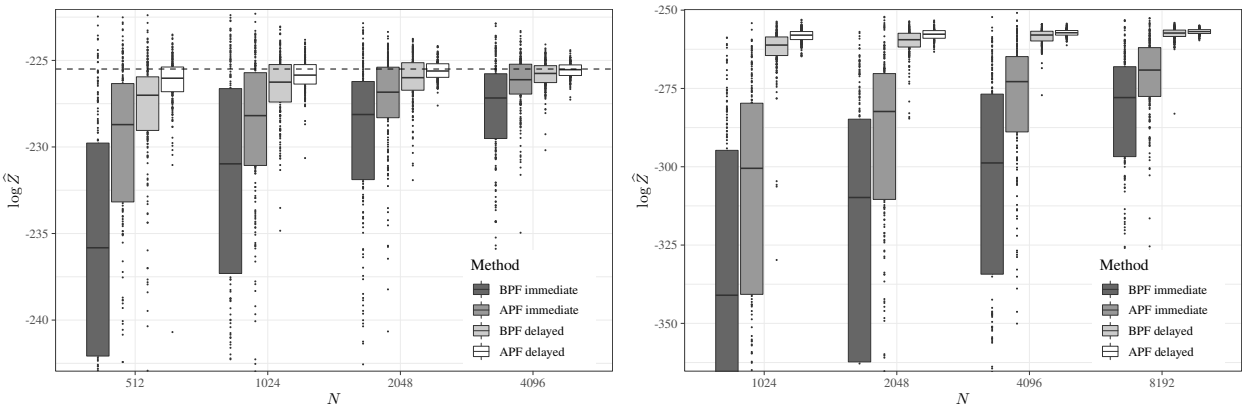
In this paper we introduced a new general inference method for probabilistic programming combining an extended alive particle filter (APF) with delayed sampling, and proved that the resulting estimate of the marginal likelihood is unbiased. We showed how phylogenetic birth-death models can be implemented in probabilistic programming languages, in particular, we considered two models—CRBD and BiSSE and their implementation in the probabilistic programming language Birch. We showed the strength of this inference method for these models compared to the standard bootstrap particle filter (BPF) (Tables 1 and 2, and Figures 3a and 3b): for the BiSSE model using 8192 particles we increased RESS approximately 29 times, CAR approximately 30 times and lowered $\operatorname{var} \log \widehat{Z}$ more than 1150 times at the cost of running 3 times more propagations.

The extended APF is a suitable drop-in replacement for the BPF for black-box probabilistic programs. If a program produces only positive weights, the APF produces the same result as the BPF at the overhead of just one extra particle, used to estimate the marginal likelihood. On the other hand, if the program can produce zero weights, the APF behaves much more reasonably than the BPF: resampling and propagation are repeated until all particles have positive weight. This may seem equivalent to using the BPF with a higher number of particles (ρ times more to be precise), but this is not the case: the number of propagations P_t is not the same throughout the execution, but rather adapted dynamically for each t. This simplifies the tuning of the number of particles for such models.

The learning of rates in birth-death models sits in the context of broader problems in phylogenetics, such as the learning of trees. Interesting future work would be to consider whether models and methods for learning rates can be combined with models and methods for learning tree structures for joint inference.

Acknowledgements

The authors wish to thank Johannes Borgström, Viktor Senderov and Andreas Lindholm for their useful feedback. This research was financially supported by the Swedish Foundation for Strategic Research (SSF) via the project ASSEMBLE and by the Swedish Research Council grants 2013-4853, 2014-05901 and 2017-03807.



(a) CRBD

(b) BiSSE

Figure 3: Box plot of $\log \widehat{Z}$ for the CRBD (on the left) and BiSSE (on the right) model for different number of particles (N) and methods. The lower and upper hinges correspond to the first and third quartile, whereas the midhinge corresponds to the median. Values outside of the interquartile range are shown as dots. The horizontal dashed line shows the true value of $\log Z$ for the CRBD model.

Table 1: Summary of the results of the experiments with the CRBD model using the cetacean phylogeny as the observation, priors $\lambda, \mu \sim \text{Gamma}(1, 1)$, and $M = 200$.

	Bootstrap particle filter (BPF)						**Alive particle filter (APF)**							
	Immediate sampling			Delayed sampling			Immediate sampling				Delayed sampling			
N	RESS	CAR	var	RESS	CAR	var	ρ	RESS	CAR	var	ρ	RESS	CAR	var
512	0.02	0.04	334.2	0.13	0.23	31.6	1.8	0.11	0.15	50.7	1.7	0.40	0.46	2.7
1024	0.11	0.12	117.5	0.28	0.35	12.9	1.8	0.14	0.18	20.2	1.7	0.54	0.55	0.8
2048	0.14	0.17	52.2	0.47	0.47	8.7	1.7	0.29	0.32	7.6	1.7	0.73	0.69	0.3
4096	0.18	0.23	17.2	0.67	0.63	0.7	1.7	0.36	0.42	2.7	1.7	0.84	0.76	0.2

Table 2: Summary of the results of the experiments with the BiSSE model using the cetacean phylogeny extended with information about the body length as the observation, priors $\lambda_0, \lambda_1, \mu_0, \mu_1 \sim \text{Gamma}(1, 1)$ and $\varsigma \sim \text{Gamma}(1, 10/820.28)$, and $M = 200$.

	Bootstrap particle filter (BPF)						**Alive particle filter (APF)**							
	Immediate sampling			Delayed sampling			Immediate sampling				Delayed sampling			
N	RESS	CAR	var	RESS	CAR	var	ρ	RESS	CAR	var	ρ	RESS	CAR	var
1024	0.01	0.01	3382.2	0.06	0.09	72.4	10.0	0.01	0.01	2294.9	3.1	0.10	0.21	4.8
2048	0.01	0.01	2954.0	0.09	0.15	22.2	6.6	0.02	0.02	1044.5	3.1	0.14	0.27	2.9
4096	0.01	0.01	1894.1	0.22	0.27	7.6	5.9	0.01	0.01	614.3	3.1	0.34	0.43	1.3
8192	0.02	0.02	968.4	0.28	0.35	6.1	3.9	0.02	0.03	160.9	3.0	0.54	0.55	0.8

References

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019.

R. Bouckaert, J. Heled, D. Kühnert, T. Vaughan, C.-H. Wu, D. Xie, M. A. Suchard, A. Rambaut, and A. J. Drummond. BEAST 2: A software platform for Bayesian evolutionary analysis. *PLOS Computational Biology*, 10(4):e1003537, 2014.

B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1), 2017.

P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

P. Del Moral, A. Jasra, A. Lee, C. Yau, and X. Zhang. The alive particle filter and its use in particle Markov chain Monte Carlo. *Stochastic Analysis and Applications*, 33(6):943–974, 2015.

A. J. Drummond and A. Rambaut. BEAST: Bayesian evolutionary analysis by sampling trees. *BMC Evolutionary Biology*, 7(1):214, 2007.

H. Ge, K. Xu, and Z. Ghahramani. Turing: A language for flexible probabilistic inference. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, pages 1682–1690, 2018.

N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 220–229, 2008.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings on Radar and Signal Processing*, volume 140, pages 107–113, 1993.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57 (1):97–109, 1970.

S. Höhna, M. J. Landis, T. A. Heath, B. Boussau, N. Lartillot, B. R. Moore, J. P. Huelsenbeck, and F. Ronquist. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology*, 65(4): 726–736, 2016.

J. P. Huelsenbeck and F. Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8): 754–755, 2001.

D. G. Kendall. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics*, 19(1): 1–15, 1948.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

W. P. Maddison, P. E. Midford, and S. P. Otto. Estimating a binary character's effect on speciation and extinction. *Systematic Biology*, 56(5):701–710, 2007.

V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

L. M. Murray and T. B. Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43, 2018.

L. M. Murray, E. M. Jones, and J. Parslow. On disturbance state-space models and the particle marginal metropolis-hastings sampler. *SIAM/ASA Journal on Uncertainty Quantification*, 1(1):494–521, 2013.

L. M. Murray, D. Lundén, J. Kudlicka, D. Broman, and T. B. Schön. Delayed sampling and automatic Rao-Blackwellization of probabilistic programs. In *International Conference on Artificial Intelligence and Statistics*, pages 1037–1046, 2018.

F. F. Nascimento, M. dos Reis, and Z. Yang. A biologist's guide to Bayesian phylogenetic analysis. *Nature Ecology & Evolution*, 1(10):1446–1454, 2017.

R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

B. Paige and F. Wood. A compilation target for probabilistic programming languages. In *International Conference on Machine Learning*, pages 1935–1943, 2014.

D. L. Rabosky and E. E. Goldberg. Model inadequacy and mistaken inferences of trait-dependent speciation. *Systematic Biology*, 64(2):340–355, 2015.

G. J. Slater, S. A. Price, F. Santini, and M. E. Alfaro. Diversity versus disparity and the radiation of modern cetaceans. *Proceedings of the Royal Society of London B: Biological Sciences*, 277(1697):3097–3104, 2010.

M. E. Steeman, M. B. Hebsgaard, R. E. Fordyce, S. Y. Ho, D. L. Rabosky, R. Nielsen, C. Rahbek, H. Glenner, M. V. Sørensen, and E. Willerslev. Radiation of

extant cetaceans driven by restructuring of the oceans. *Systematic Biology*, 58(6):573–585, 2009.

D. Tolpin, J.-W. van de Meent, H. Yang, and F. Wood. Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, pages 6:1–6:12. ACM, 2016.

D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.