

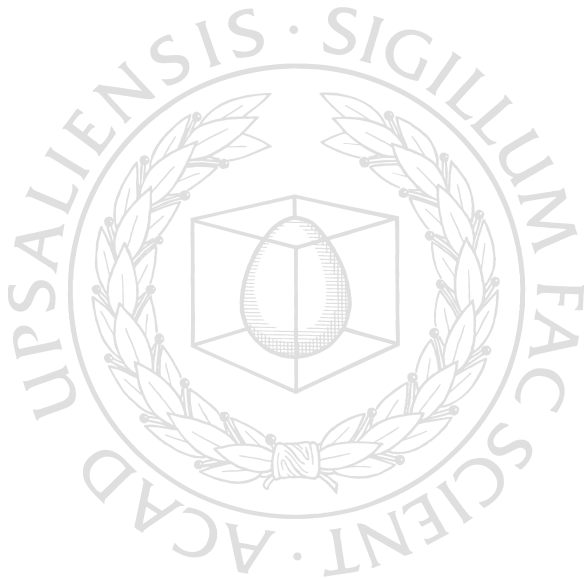


UPPSALA
UNIVERSITET

*Digital Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology 2006*

Probabilistic Programming for Birth-Death Models of Evolution

JAN KUDLICKA



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2021

ISSN 1651-6214
ISBN 978-91-513-1123-4
urn:nbn:se:uu:diva-432409

Dissertation presented at Uppsala University to be publicly examined in ITC 2446, Lägerhyddsvägen 2, Uppsala, Thursday, 25 March 2021 at 14:00 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Bret Larget (University of Wisconsin).

Abstract

Kudlicka, J. 2021. Probabilistic Programming for Birth-Death Models of Evolution. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 2006. 84 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-513-1123-4.

Phylogenetic birth-death models constitute a family of generative models of evolution. In these models an evolutionary process starts with a single species at a certain time in the past, and the speciations—splitting one species into two descendant species—and extinctions are modeled as events of non-homogenous Poisson processes. Different birth-death models admit different types of changes to the speciation and extinction rates.

The result of an evolutionary process is a binary tree called a phylogenetic tree, or phylogeny, with the root representing the single species at the origin, internal nodes speciation events, and leaves currently living—extant—species (in the present time) and extinction events (in the past). Usually only a part of this tree, corresponding to the evolution of the extant species and their ancestors, is known via reconstruction from e.g. genomic sequences of these extant species.

The task of our interest is to estimate the parameters of birth-death models given this reconstructed tree as the observation. While encoding the generative birth-death models as computer programs is easy and straightforward, developing and implementing bespoke inference algorithms are not. This complicates prototyping, development, and deployment of new birth-death models.

Probabilistic programming is a new approach in which the generative models are encoded as computer programs in languages that include support for random variables, conditioning on the observed data, as well as automatic inference. This thesis is based on a collection of papers in which we demonstrate how to use probabilistic programming to solve the above-mentioned task of parameter inference in birth-death models. We show how these models can be implemented as simple programs in probabilistic programming languages. Our contribution also includes general improvements of the automatic inference methods.

Keywords: probabilistic programming, birth-death models, statistical phylogenetics, particle filters, sequential Monte Carlo (SMC)

Jan Kudlicka, Department of Information Technology, Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

© Jan Kudlicka 2021

ISSN 1651-6214

ISBN 978-91-513-1123-4

urn:nbn:se:uu:diva-432409 (<http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-432409>)

*“Mathematics reveals its secrets only to those
who approach it with pure love, for its own beauty.”*

— Archimedes

List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I F. Ronquist[†], J. Kudlicka[†], V. Senderov[†], J. Borgström, N. Lartillot, D. Lundén, L. Murray, T. B. Schön, and D. Broman. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. Preprint at <https://www.biorxiv.org/content/10.1101/2020.06.16.154443v4>, 2020.

[†] Equal contribution.

II J. Kudlicka, L. M. Murray, F. Ronquist, and T. B. Schön. Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Tel Aviv, Israel, 2019.

III 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 (AISTATS)*, pages 1037–1046, Playa Blanca, Lanzarote, Spain, 2018.

IV J. Kudlicka, L. M. Murray, T. B. Schön, and F. Lindsten. Particle filter with rejection control and unbiased estimator of the marginal likelihood. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5860–5864, 2020. © 2019 IEEE. Reprinted with permission from the authors.

Reprints were made with permission from the respective copyright holders.

Contributions

Paper [I]

The ideas, concepts and algorithms were developed by all authors in close collaboration. The algorithms and models were implemented by me, V. Senderov and F. Ronquist. Verification experiments and empirical analyses were run by me and V. Senderov. We together also generated most of the illustrations. All authors contributed to writing and revising the manuscript and the supplementary material.

Paper [II]

The ideas were developed in close collaboration among all authors. I implemented the models and support for delayed sampling for relevant distributions in Birch, and contributed to implementing, testing and debugging the extended alive particle filter. I ran all experiments and analyses, and wrote the manuscript, including the proof of unbiasedness of the marginal likelihood estimator for the extended alive particle filter.

Paper [III]

The methods and algorithms used in delayed sampling were developed in close collaboration among all authors. In addition I contributed to developing the examples, and prepared Figure 1 in the manuscript. I also gave detailed feedback on manuscript drafts.

Paper [IV]

The ideas are based on unpublished work that was done in collaboration with L. M. Murray and T. B. Schön, and on [II]. The idea for the first use case was developed in discussion with F. Lindsten, and the second use case is based on a model previously developed by L. M. Murray. The models were implemented in Birch by me and L. M. Murray. I ran all experiments and analyses, and wrote the manuscript, including the proofs.

Contents

Introduction	9
1 Probabilistic programming	11
1.1 Introduction	11
1.2 Basic concepts	16
1.2.1 Defining and using random variables	16
1.2.2 Conditioning on the observed data	18
1.2.3 Automatic inference	19
1.2.4 Illustrative examples	20
1.2.5 Existing probabilistic programming languages	22
1.3 Programmatic model	22
1.4 Sequential Monte Carlo based inference	24
1.4.1 Introduction	24
1.4.2 Monte Carlo integration	24
1.4.3 Rejection sampling	25
1.4.4 Importance sampling	25
1.4.5 Importance sampling for state-space models	26
1.4.6 Sequential Monte Carlo	29
2 Birth-death models of evolution	33
2.1 Introduction	33
2.2 Complete and surviving trees	34

2.3	Constant-rate birth-death model	37
2.4	Selected non-constant-rate birth-death models	39
2.4.1	Time-dependent birth-death models	39
2.4.2	Lineage-specific birth-death-shift model	40
2.4.3	Bayesian analysis of macro-evolutionary mixtures	40
2.4.4	Cladogenetic diversification rate shift models	41
2.4.5	Birth-death models with state	42
2.5	Bayesian inference of the model parameters	43
2.6	Likelihood function for the CRBD model	44
3	Probabilistic programming for phylogenetics	49
3.1	Parameter inference for the CRBD model	49
3.2	Alive particle filter	54
3.3	Delayed sampling	57
3.4	Conditioning on the time of the most recent common ancestor	63
4	Conclusion	67
5	Acknowledgments	69
6	Summary in Swedish	71
A	Used distributions and their parameterizations	79
B	Used abbreviations	83

Introduction

Welcome to the exciting world of probabilistic programming and phylogenetics!

Phylogenetic birth-death models are a family of rather simple models of evolution of species (or other taxonomic groups). Starting with a single species in the past, they essentially model *speciation* events, when a parent species splits into two descendant species, and *extinction* events, when the whole population of a species dies out. Biologists use these models to estimate their parameters (such as the speciation and extinction rates) based on a phylogenetic tree representing the evolution of the currently living, *extant*, species. This tree can be inferred from our knowledge about some of the extant species, e.g. their morphological traits and genomic data. Estimation of the parameters is challenging due to the fact that this tree is usually only a part of a complete tree that also includes the evolution of (unknown) extinct species.

Birth-death models of evolution play an important role in statistical phylogenetics, yet the path from a new model idea to a software implementation ready to be used by biologists is time-consuming and error-prone. Describing a new model in the language of mathematics, deriving a bespoke inference algorithm, and then implementing it, either in an existing software package, or creating a new one, takes a lot of time and requires experts from several different areas.

Recent developments in probabilistic programming open up an exciting alternative aiming to shorten and simplify the whole process significantly: new models can be expressed as short programs in probabilistic programming languages with just a basic understanding of programming, and without deep knowledge of the complex phylogenetic software packages or libraries. There is no need to derive any bespoke inference algorithm for a new model (nor to implement it): one of the main features of probabilistic programming languages is automatic inference. From the user's point of view, the inference should be as simple as running the program.

During the last five years my colleagues from Uppsala University, KTH Royal Institute of Technology and Swedish Museum of Natural History (as well as other collaborators from abroad) and I have been working towards this goal, not

only to demonstrate that this is indeed possible, but also to make the inference for phylogenetic models fast and efficient.













This thesis is based on a collection of papers written during this exciting time. It also includes a few introductory chapters: Chapter 1 (p. 11) and Chapter 2 (p. 33) introduce probabilistic programming and birth-death models of evolution. Chapter 3 (p. 49) links both topics together and demonstrates how probabilistic programming and probabilistic programming languages can be used in phylogenetics. It also provides a common thread connecting the papers. The conclusion, including some ideas for the future work, is the contents of Chapter 4 (p. 67).

Probabilistic programming

1.1 Introduction

Let us start with a simple experiment: Take a die, roll it twice and add the faces. The outcome of both throws, which we will denote by X_1 and X_2 , are random variables, and so is their sum $S = X_1 + X_2$. It makes perfect sense to ask questions like these: What is the probability that this sum is 2? Or 3? Or any other possible value? Or said differently, what is the probability distribution of the sum of the faces?

If you are reading this thesis, you probably know quite a lot about probability and statistics, and might already have an idea about how to answer the questions above. In either case, let us go through the solution together. The table below shows all possible outcomes of X_1 and X_2 and their corresponding sum S . We will assume that the die is fair, i.e., the probability of each face is the same. To answer our questions, we just need to count the fractions of each possible value of S in the table.

		Outcome of X_1					
							
Outcome of X_2		2	3	4	5	6	7
		3	4	5	6	7	8
		4	5	6	7	8	9
		5	6	7	8	9	10
		6	7	8	9	10	11
		7	8	9	10	11	12

For example, the probability of the sum being 8 (shown in bold) is $5/36$ as there are 5 combinations where $S = 8$ (namely (6, 2), (5, 3), (4, 4), (3, 5) and (2, 6))

out of the 36 possible combinations. If we do this for all values of the sum we get the distribution we are interested in:

s	2	3	4	5	6	7	8	9	10	11	12
$P(S = s)$	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$

This was quite a simple experiment, but how would we go around if we wanted a computer to (automatically) determine this distribution? It is rather easy to write a function (procedure, program) that can simulate the experiment in any programming language that supports random numbers. For example, in Python, such a function could look like this:

```
import random

def model():
    die1 = random.randint(1, 6)
    die2 = random.randint(1, 6)
    return die1 + die2
```

Can we somehow use this function to calculate (an estimate of) the probability distribution? We can call the function we just created “a huge number” of times, and count how many times the function returns 2, 3, . . . , 12:

```
def main():
    N = 10**5
    dist = {}
    for i in range(N):
        s = model()
        if s not in dist:
            dist[s] = 0.0
        dist[s] += 1/N
    return dist
```

In the main function we called the `model` function 10^5 times and counted the relative frequencies of all values returned from the function. Figure 1.1 shows the resulting distribution as a bar plot, along with the true probabilities for all values of the sum.

The approach of repeating an experiment to obtain an estimate of the probability distribution of interest is known as the *Monte Carlo* method and it forms the foundation for almost all methods we will use and develop in this thesis.

Now, let us make the experiment a bit more complicated. The die is rolled by a friend and we can't see the outcomes. After the experiment we are told that the sum is less than 5. What can we say about the distribution of X_1 conditioned on this fact?

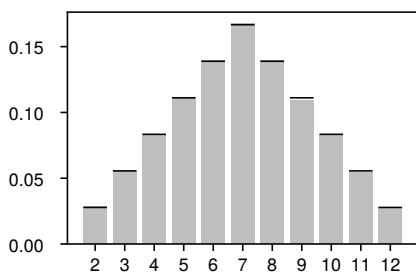


Figure 1.1: Result of the dice experiment. The gray bars represent the estimated probabilities, the black lines the corresponding true probabilities.

Again, we can look at all possible combinations and eliminate those where the sum S is greater than or equal to 5 (shown in gray):

		Outcome of X_1					
Outcome of X_2		2	3	4	5	6	7
		3	4	5	6	7	8
		4	5	6	7	8	9
		5	6	7	8	9	10
		6	7	8	9	10	11
		7	8	9	10	11	12

There are only 6 combinations that meet the observation, three of them with $X_1 = 1$, two with $X_1 = 2$, and one with $X_1 = 3$, leading to the following conclusion:

x_1	1	2	3	4	5	6
$P(X_1 = x_1 S < 5)$	$\frac{3}{6}$	$\frac{2}{6}$	$\frac{1}{6}$	0	0	0

Before we return to the computer simulations, let us think a little bit about what is happening while we are observing our friend do the experiment. Before we get any information, we believe that the probability of each possible outcome of X_1 is $1/6$ —that is our prior belief. As soon we get to know that the sum is less than 5, our belief about (the probability distribution of) X_1 changes. Well, our brains might not immediately calculate the exact probabilities, but we can immediately understand that 4, 5 and 6 are impossible outcomes, i.e., $P(X_1 = 4 | S < 5) = P(X_1 = 5 | S < 5) = P(X_1 = 6 | S < 5) = 0$.

The process of updating the *prior* belief (that is, the belief before the observation) to the *posterior* belief (the belief after the observation) is called *inference*. We looked at a very simple example with two dice, but in general, the inference

for realistic problems might be rather difficult. Developing inference algorithms is a time-consuming and error-prone process. It would be convenient if we could just implement a *generative model*, i.e. a model that just simulates the experiment, and then add statements about the observations, and run the program to get the posterior distribution of interest (or some of its statistics).

Returning to our experiment with the dice, we would like to extend the `model` function we wrote by just adding a single line stating that “the sum is less than 5” (and returning the outcome of the throw instead of the sum):

```
def model():
    die1 = random.randint(1, 6)
    die2 = random.randint(1, 6)
    observe(die1 + die2 < 5)
    return die1
```

Probabilistic programming languages (PPLs) allows us to do exactly this. They extend general purpose (deterministic) programming languages and provide

- support for random variables,
- conditioning on the observed data, and
- automatic inference.

In the existing PPLs the support for random variables is usually more *ergonomic* than what we have seen in our code. For example, instead of

```
die1 = random.randint(1, 6)
```

we can very often see something like

```
die1 ~ DiscreteUniform(1, 6)
```

From the user’s point of view, a recipe for probabilistic modeling with a PPL is very simple:

1. Write a generative model as a computer program.
2. Add observe statements.
3. Run the program to automatically do the inference.

But how does this automatic inference work? We will introduce some of the algorithms later, but to offer some intuition already now, consider the following trivial (and often rather inefficient) algorithm, known as *rejection sampling* [62]. Again, we are going to call the `model` function many times, but in the case when the boolean condition in the observe statement is not true, we

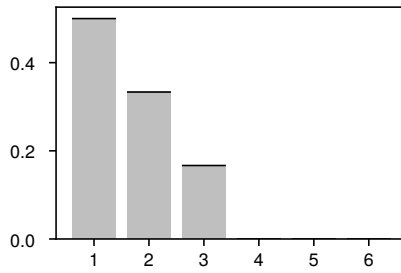


Figure 1.2: Result of the dice experiment with conditioning on the sum being less than 5. The gray bars represent the estimated posterior probabilities, the black lines the corresponding true probabilities.

will return immediately and repeat the function call (and keep repeating until the condition is eventually true). Python is not a probabilistic programming language, but we can use (or rather misuse) exceptions to implement rejection sampling:

```
class RejectionException(Exception):
    pass

def observe(cond):
    if not cond:
        raise RejectionException

def main():
    N = 10**5
    dist = {}
    for i in range(N):
        while True:
            try:
                s = model()
                break
            except RejectionException:
                pass
        if s not in dist:
            dist[s] = 0.0
        dist[s] += 1/N
    return dist
```

Running this program for our toy model results in a distribution shown in Figure 1.2.

Such an approach obviously works only if the rejection ratio is fairly small. We also need to be able to work with continuous random variables, since, after all, most of the random variables in realistic problems are indeed continuous.

In the next section we will define probabilistic programming and PPLs more

formally, go through a few more sample models, and show how they can be implemented as simple programs in PPLs. Execution of probabilistic programs can be modeled using a *programmatic model*, which is introduced in Section 1.3 (p. 22). In Section 1.4 (p. 24) we will turn our attention to automatic inference and the algorithms supporting it, and go deeper into the sequential Monte Carlo (SMC) based inference.

1.2 Basic concepts

As we have seen in the previous section, probabilistic programming is a new programming paradigm that allows encoding a probabilistic model as a computer program, and running the inference automatically, without the need to implement a bespoke inference algorithm.

Probabilistic programming languages (PPLs) can be seen as extensions of general-purpose deterministic programming languages with probabilistic constructs (such as defining random variables and conditioning on the observed data) and automatic inference.

1.2.1 Defining and using random variables

By defining a random variable we inform the compiler or the interpreter about the variable's identifier (i.e., the name of the random variable) and the probability distribution of the random variable. We will use the following notation in pseudocode:

$$x \sim D(\theta_1, \theta_2, \dots)$$

Here, x denotes a random variable specified by a probability distribution D and its parameters $\theta_1, \theta_2, \dots$. Note that the parameters might, and often will, be expressions using other random variables defined in the program earlier. It is important to realize that the distribution and the parameters specify the conditional probability distribution of the random variable x given the previously defined random variables.

How do PPLs store and work with random variables? The simplest way is to represent them by random variates (i.e., particular outcomes of the random variables). In our illustrative examples in the previous section, we drew outcomes of both throws from `DiscreteUniform(1, 6)` and stored these (integer) values in the (Python) variables `die1` and `die2`. When returning the sum `die1 + die2`, the program just used these stored variate values to calculate and return the sum. Some PPLs even require the user to draw, or *sample*, a value from

the distribution and to store it in a variable (representing the random variate) explicitly. For example, consider the following snippet in WebPPL [23], a simple JavaScript-based PPL that can even execute probabilistic programs in a web browser¹:

```
var dist = Gaussian({mu: 0, sigma: 1})
var x = sample(dist)
```

Other PPLs deal with random variables in a more advanced way. For example, Birch [46], an object-oriented PPL transpiling to C++, represents random variables as objects with attributes for the specified distribution and its parameters. Birch avoids sampling a particular value as long as it is possible by employing a technique called *delayed sampling* [III]. We will return to it later, for now, consider the following program demonstrating an interesting case where the value of x is never even sampled:

```
x:Random<Real>;
y:Random<Real>;

x ~ Gaussian(0, 1);
y ~ Gaussian(x, 1);
stdout.print("y = " + y.value() + "\n");
```

When executing the program, Birch automatically exploits the conjugacy relationship between $p(x) = \mathcal{N}(0, 1)$ and $p(y|x) = \mathcal{N}(x, 1)$. When the value of y is needed in order to print it to standard output, it is sampled from the marginal distribution $p(y) = \mathcal{N}(0, 2)$. The distribution related to x is then updated based on the sampled value of y , but since we do not need the value of x , it will never get sampled.

Probabilistic programs can use the random variables as any other (deterministic) variables, including to control the flow of the execution in conditionals, loops and recursion. Let us have a look at two examples; the first example demonstrates using a random variable in the predicate of a conditional:

```
x ~  $\mathcal{N}(0, 1)$ 
if x > 0 then
  y ~ Exponential(x)
else
  y ~ Uniform(0, 1)
end if
```

The second example demonstrates how to use unbounded recursion to define the geometric distribution:

¹<http://webppl.org>

```

function GEOMETRIC( $p$ )
   $x \sim \text{Bernoulli}(p)$ 
  if  $x$  then
    return 1
  else
    return 1 + GEOMETRIC( $p$ )
  end if
end function

```

Stochastic branching and unbounded recursion make computationally universal PPLs more expressive than probabilistic graphical models (PGM) [32]: all models that can be expressed in the PGM framework can also be expressed using computationally universal, or Turing-complete, PPLs, while the opposite is not true.

1.2.2 Conditioning on the observed data

Probabilistic programming is closely related to Bayesian probabilistic modeling: some random variables are observed and we want to reason about the (posterior) probability distribution of the remaining—unobserved or latent—random variables given the values of the observed variables.

This is where the *observe* construct comes into play. Both the observed value and the probability distribution of the observed random variable must be specified. In pseudocode we will use the following notation:

```

observe  $y \sim D(\theta_1, \theta_2, \dots)$ 

```

As an example, consider a simple linear Gaussian state-space model encoded as a probabilistic program:

```

 $x_0 \sim \mathcal{N}(0, 0.1)$ 
for  $t \in \{1, 2, \dots, 100\}$  do
   $x_t \sim \mathcal{N}(0.5x_{t-1}, 0.1)$ 
  observe  $y_t \sim \mathcal{N}(0.7x_t, 0.1)$ 
end for

```

Here, x_t denotes the unknown (unobserved, latent) state at time t , and y_t the corresponding measurement.

Before we move on to the automatic inference, recall the example from the first section, where we used `observe(die1 + die2 < 5)`. We did not specify any distribution here; we just used a simple predicate. This is however equivalent to

observe true \sim Bernoulli([die1 + die2 < 5])

The parameter is specified using the Iverson bracket $[P]$:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

1.2.3 Automatic inference

Let X denote all latent random variables in our model, Y all observed random variables, and y the observed values.

For most problems, to implement a generative program (encoding the joint distribution $p(X, Y)$) is simple. As we have seen above, we do not even need PPLs to do so; any programming language with support for drawing random numbers will suffice. We can run such a program multiple times to simulate the model and draw samples from the joint distribution. We can then use these samples to estimate the distribution or the expected value of any test function with respect to this distribution.

However, changing the program to be able to sample from the posterior distribution $P(X|Y = y)$ in standard programming languages might be rather difficult and involves finding (and possibly tailoring) a suitable inference algorithm or, if we are unlucky, developing and implementing a completely new bespoke inference algorithm.

In PPLs such a change is trivial. For example, consider a potentially biased coin with unknown probability of heads π , and an experiment in which we throw the coin 50 times and count the number of heads, encoded as the following generative program:

```
 $\pi \sim \text{Uniform}(0, 1)$   
 $c \sim \text{Binomial}(50, \pi)$ 
```

Now, how can we get the posterior distribution of π given that we have seen 28 heads? In PPLs, we only need to change the definition of the random variable c to an observe:

```
 $\pi \sim \text{Uniform}(0, 1)$   
observe 28  $\sim \text{Binomial}(50, \pi)$ 
```

Running this program produces a sample from the posterior distribution. The inference is automatic; we have not implemented any inference algorithm anywhere in the program. This decoupling of the model and the inference is

one of the main advantages of PPLs. Although it might sound like magic, automatic inference is based on composing suitable general-purpose inference algorithms, some of which we will cover later in this chapter.

Extending and developing new general inference algorithms and heuristics to make automatic inference better (in terms of speed and variance of the estimated quantities) are the subject of ongoing research.

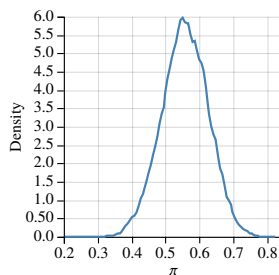
1.2.4 Illustrative examples

Let us now look at a couple of models and their implementation in WebPPL [23]. First, let us return to the example with a coin from the previous paragraph:

```
var coinExperiment = function() {  
  var  $\pi$  = sample(Uniform({a: 0, b: 1}))  
  observe(Binomial({n: 50, p:  $\pi$ }), 28)  
  return  $\pi$   
}
```

```
Infer({model: coinExperiment, method: 'MCMC', samples: 100000})
```

The model itself is implemented as a function (`coinExperiment`) which is passed together with the inference parameters (we use Markov chain Monte Carlo (MCMC) as the inference method to collect 100000 samples) to the `Infer` function. This function is responsible for running the inference and produces the estimate of the posterior distribution for π (note the return statement at the end of the `coinExperiment` function). If we run the program in a web browser, WebPPL will also visualize the distribution:



In the second and more advanced example we will demonstrate how easy it is to implement Bayesian linear regression in WebPPL. Consider the following model:

$$\beta_0 \sim \mathcal{N}(0, 1),$$

$$\beta_1 \sim \mathcal{N}(0, 1),$$

$$y_n \sim \mathcal{N}(\beta_0 + \beta_1 x_n, 0.01)$$

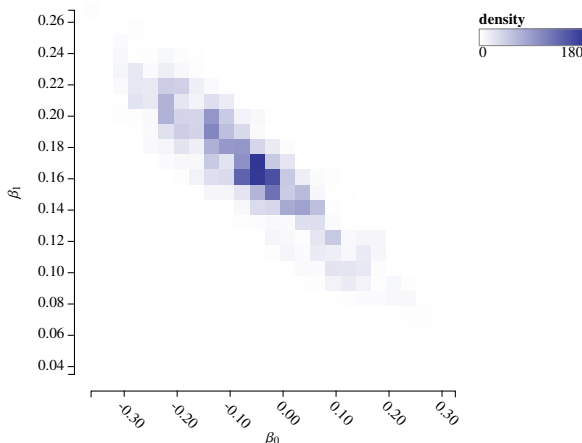
and the following measurements:

x_n	1	2	3	4	5
y_n	0.1993	0.1401	0.4304	0.6206	0.7807

The implementation of this model is straightforward, perhaps with the exception of using the `map` function to process the measurements (WebPPL does not support for-loops):

```
var observation = [  
  [1, 0.1993],  
  [2, 0.1401],  
  [3, 0.4304],  
  [4, 0.6206],  
  [5, 0.7807]  
]  
  
var regression = function() {  
  var  $\beta_0$  = sample(Gaussian({mu: 0, sigma: 1}))  
  var  $\beta_1$  = sample(Gaussian({mu: 0, sigma: 1}))  
  map(function(xy) {  
    var x = xy[0], y = xy[1]  
    observe(Gaussian({mu:  $\beta_0$  +  $\beta_1$ *x, sigma: 0.1}), y)  
  }, observation)  
  return [ $\beta_0$ ,  $\beta_1$ ]  
}  
  
viz.heatMap(Infer({  
  model: regression, method: 'SMC', particles: 20000  
}))
```

We use SMC-based inference (with 20000 particles) and visualize the distribution of the parameters β_0, β_1 in the form of a heat map:



1.2.5 Existing probabilistic programming languages

We have so far seen a few examples of probabilistic programs in Birch and WebPPL. As with deterministic programming languages, there exist many PPLs to choose from, including the following languages (in alphabetical order): Anglican [59], BayesDB [37], Biips [58], Birch [46], BLOG [40], BUGS [21], Church [24], Edward [60], Figaro [52], Gen [10], Hakaru [49], JAGS [53], LibBi [44], Probabilistic-C [51], Pyro [5], STAN [8], Turing.jl [19], Venture [38], WebPPL [23].

These probabilistic programming languages support various programming paradigms, implement various automatic inference algorithms, and target various scientific communities. Many of them are based on existing deterministic languages and/or use existing libraries for machine learning.

For example, Birch, which most of the work included in this thesis is related to, is an object oriented PPL transpiling to C++, and implementing several sequential Monte Carlo (SMC) based inference algorithms.

1.3 Programmatic model

Before we look closer at some of the inference algorithms, let us describe an execution model of probabilistic programs, based on the programmatic model by Murray and Schön [46].

Let $\{V_i\}$ denote a set of all (both latent and observed) random variables in a probabilistic program. Note that this set is either finite or countably infinite. In general we cannot make a one-to-one map from the definitions of random variables and observes in the program to the variables in $\{V_i\}$. Consider, for example, random variables in a loop or in a function that is called multiple times. A definition of a random variable at a given location in the program might correspond to multiple random variables in $\{V_i\}$.

When a program is executed, it encounters the random variables in a specific order. This order might be different for different executions due to stochastic branching (with the exception of the very first encountered random variable). Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_{|\sigma|})$ denote a sequence of indices into the set $\{V_i\}$ in which the random variables are encountered during the execution. Further, let v_i denote a realization of the encountered random variable V_i . If V_i has not been encountered during the execution, the corresponding variate v_i has no value, which we will denote by $v_i = \perp$.

We can define the execution state as a set $x = \{(i, v_i)\}_{i \in \sigma}$ of tuples consisting of indices of the variables encountered so far and their corresponding variates.

At any time during the execution, the index κ of a random variable to be encountered next depends on the current state x deterministically, i.e.

$$\kappa = \text{Ne}(x),$$

where Ne (for *next*) denotes this deterministic function. In case there are no more random variables to be encountered, the Ne function will return \perp .

Also the parameters of the distribution p_κ associated with the corresponding random variable V_κ (and given by the probabilistic program) depend on the state deterministically, i.e.

$$V_\kappa \sim p_\kappa(\text{Pa}(x)),$$

where Pa (for *parents*) denotes this deterministic function.

Once V_κ has been realized, κ is appended to the sequence σ , and the current state is updated to $x \cup \{(\kappa, v_\kappa)\}$.

The joint probability distribution of yet unencountered random variables, given the current state can be stated recursively [II]:

$$p(\{v_i\}_{i \notin x} | x) = \begin{cases} p_\kappa(v_\kappa | \text{Pa}(x)) \times p(\{v_i\}_{i \notin x \cup \{(\kappa, v_\kappa)\}} | x \cup \{(\kappa, v_\kappa)\}) & \text{if } \kappa \neq \perp, \\ 1 & \text{if } \kappa = \perp \wedge \forall i \notin x : v_i = \perp, \\ 0 & \text{otherwise,} \end{cases}$$

where $\kappa = \text{Ne}(x)$ and $i \in x$ (resp. $i \notin x$) means that there exists (resp. does not exist) a pair in x whose first element is i . The first case is an application of the chain rule, while the second and third case cover the situation where there are no more random variables to encounter (in that case each member of $\{v_i\}_{i \notin x}$ must be \perp). The joint distribution of all random variables encoded by a probabilistic program is given by $p(\{v_i\} | \emptyset)$.

Let $\gamma = (\gamma_1, \gamma_2, \dots, \gamma_T)$ denote a sequence of indices of the observed variables corresponding to the observations y_1, y_2, \dots, y_T . The goal of the inference is to determine (estimate) the posterior distribution

$$p(\{v_i\}_{i \notin \gamma} | \{(\gamma_t, y_t)\}_{t=1}^T).$$

We will assume that each execution encounters all observed variables and in the same order (given by γ), and, for the sake of keeping the algorithms simple, that the last observed variable is also the very last encountered random variable.

1.4 Sequential Monte Carlo based inference

1.4.1 Introduction

Automatic inference in PPLs is based on inference algorithms that were originally developed for other types of models (for example, state-space models). Exact inference algorithms, such as Kalman filtering [30] or enumeration (iterating through all possible outcomes), are methods suitable only for a small class of programs.

On the other hand, approximate and evaluation based inference methods, such as Monte Carlo methods and variational inference (e.g. [61, 63]) as well as combinations of these methods (e.g. [48]), are easily applicable to a much larger class of programs. Monte Carlo methods cover a huge family of algorithms, including Metropolis–Hastings algorithm [39, 26], pseudo-marginal Metropolis–Hastings algorithm [1], Gibbs sampling [20], Hamiltonian Monte Carlo (HMC) [50], and sequential Monte Carlo (SMC) [14, 13].

1.4.2 Monte Carlo integration

Monte Carlo integration (e.g. [7]) is a method of estimating the expected value of a test function $h(x)$ of a random variable² $x \sim p(x)$, i.e.

$$I = \mathbb{E}_p[h(x)] = \int h(x)p(x)dx.$$

Under the assumption that we can sample from $p(x)$ we can draw N samples $\{x^n\}_{n=1}^N$, and estimate the expected value using the following estimator:

$$\widehat{I}_N = \frac{1}{N} \sum_{n=1}^N h(x^n).$$

The law of large numbers ensures that

$$\lim_{N \rightarrow \infty} \widehat{I}_N = I \text{ with probability 1,}$$

and the central limit theorem ensures that

$$\sqrt{N}(\widehat{I}_N - I) \rightarrow \mathcal{N}(0, \sigma^2) \text{ in distribution,}$$

where $\sigma^2 = \text{var}_p h(x)$.

²Analogously to the notation used in probabilistic programs, we will usually use small letters to denote random variables from now on. We will also assume that continuous random variables admit probability density functions.

1.4.3 Rejection sampling

What can we do if we cannot sample from $p(x)$? Rejection sampling [62], which we have used in the first section, is one of the options we have. Assume that

- $p(x)$ can be evaluated point-wise for all x , and that
- there exists another distribution $q(x)$, which we will call the *proposal* distribution, that we can both sample from and evaluate point-wise, and such that

$$p(x) \leq Mq(x)$$

for some $M \in \mathbb{R}$ and all x .

We can get samples from $p(x)$ using the following algorithm (rejection sampling):

1. Sample x from the proposal distribution $q(x)$.
2. Sample u from the uniform distribution on $(0, 1)$.
3. If

$$u < \frac{p(x)}{Mq(x)},$$

accept x as a sample from $p(x)$, otherwise go back to step 1.

The obvious disadvantage is that it might be difficult to find a proposal $q(x)$ that ensures low rejection rate. The problem gets more prominent with increasing dimension: the rejection rate grows exponentially with the dimension of the distribution (this is sometimes referred to as the *curse of dimensionality*).

1.4.4 Importance sampling

Unlike rejection sampling, importance sampling [25] does not reject any of the proposed samples. Assume that

- we can evaluate $p(x)$ point-wise up to the proportionality, i.e. we can evaluate

$$\tilde{p}(x) = Zp(x)$$

point-wise for all x , where Z is a (possibly unknown) normalization constant, and

- there exists another distribution $q(x)$ that we can both sample from and evaluate point-wise, and such that

$$q(x) = 0 \Rightarrow p(x) = 0,$$

or in other words, that the support of $p(x)$ is a subset of the support of $q(x)$.

The expected value of $h(x)$ with respect to $p(x)$ can be rewritten as follows:

$$\mathbb{E}_p[h(x)] = \int h(x)p(x)dx = \frac{1}{Z} \int h(x) \underbrace{\frac{\tilde{p}(x)}{q(x)}}_{w(x)} q(x)dx = \frac{1}{Z} \mathbb{E}_q[h(x)w(x)].$$

The function $w(x) = \tilde{p}(x)/q(x)$ is called the *importance weight* function. The normalizing constant Z is given by

$$Z = \int \tilde{p}(x)dx = \int \frac{\tilde{p}(x)}{q(x)} q(x)dx = \int w(x)q(x)dx.$$

Both the expected value and the normalizing constant can be estimated using Monte Carlo integration. The following algorithm summarizes the estimation of the expected value of a test function using importance sampling:

1. Draw N samples $\{x^n\}_{n=1}^N$ from the proposal distribution $q(x)$.
2. Calculate the importance weights $w^n = \tilde{p}(x^n)/q(x^n)$ for $n = 1, \dots, N$.
3. Estimate the expected value as follows:

$$\mathbb{E}_p[h(x)] \approx \frac{\sum_{n=1}^N w^n h(x^n)}{\sum_{n=1}^N w^n}.$$

1.4.5 Importance sampling for state-space models

Consider a state-space model of a discrete-time stochastic process, where the state x_t at time t depends only on the state x_{t-1} at the previous time step $t - 1$, and x_t can only be observed via a random variable y_t that depends only on x_t (see the graphical representation depicted in Figure 1.3).

The joint distribution of this model is given by

$$p(x_1, x_2, \dots, x_T, y_1, y_2, \dots, y_T) = p(x_1)p(y_1|x_1) \prod_{t=2}^T p(x_t|x_{t-1})p(y_t|x_t).$$

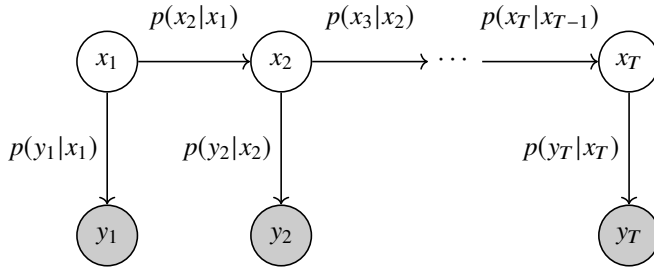


Figure 1.3: Graphical representation of a state-space model.

How can we use importance sampling to estimate the expected value of a test function with respect to the posterior distribution $p(x_1, x_2, \dots, x_T | y_1, y_2, \dots, y_T)$?

If we use

$$q(x_1, x_2, \dots, x_T) = p(x_1) \prod_{t=2}^T p(x_t | x_{t-1})$$

as the proposal distribution, the importance weight function is given by

$$w(x_1, x_2, \dots, x_T) = \frac{p(x_1)p(y_1|x_1) \prod_{t=2}^T p(x_t|x_{t-1})p(y_t|x_t)}{p(x_1) \prod_{t=2}^T p(x_t|x_{t-1})} = \prod_{t=1}^T p(y_t|x_t).$$

The form of this function allows us to calculate the importance weights in a sequential manner: for each sample $x^n = (x_1^n, x_2^n, \dots, x_T^n)$ we simulate the Markov chain, starting with sampling

$$x_1^n \sim p(x_1),$$

and then sampling x_t^n based on x_{t-1}^n in a loop:

$$x_t^n \sim p(x_t | x_{t-1}^n).$$

The corresponding importance weight can be calculated iteratively, starting with $w^n \leftarrow 1$ and updating the weight after sampling x_t^n at each time step:

$$w^n \leftarrow w^n p(y_t | x_t^n).$$

Sequential importance sampling, which is the name for importance sampling with the sequential calculation of importance weights, is one of the inference methods used in probabilistic programming languages. Recalling the programmatic model we introduced in Section 1.3 (p. 22), an execution of a

probabilistic program can be modeled as a state-space model, where the latent state x_t corresponds to the execution state at the time of processing the t -th observe, corresponding to the t -th observed random variable y_t .

Sampling from the above-mentioned proposal distribution means nothing else than running the program and sampling values of the unobserved random variables encountered during the execution. The importance weights are calculated by multiplying the likelihoods of the observed values with respect to the sampled values (realizations) of previously encountered random variables. Algorithm 1.1 summarizes sequential importance sampling for probabilistic programming. The `PROPAGATE(x)` function (Algorithm 1.2) resumes the execution of the program from the state x and continues running the program until it reaches the next observe.

Algorithm 1.1: Sequential importance sampling for probabilistic programming.

```

1: for  $n = 1, \dots, N$  do
2:    $x^n \leftarrow \emptyset$  ▷ Initialize the  $n$ -th execution of the program
3:    $w^n \leftarrow 1$ 
4: end for
5: for  $t = 1, \dots, T$  do
6:   for  $n = 1, \dots, N$  do
7:      $x^n \leftarrow \text{PROPAGATE}(x^n)$  ▷ Resume the  $n$ -th execution until it reaches next observe
8:      $w^n \leftarrow w^n p_{\gamma_t}(y_t | \text{Pa}(x^n))$ 
9:      $x^n \leftarrow x^n \cup \{(\gamma_t, y_t)\}$ 
10:  end for
11: end for
12: for  $n = 1, \dots, N$  do
13:    $h^n \leftarrow h(x^n)$  ▷ Calculate the value of the test function of interest
14: end for
15:  $\mathbb{E}[h] \approx \sum_n w^n h^n / \sum_n w^n$ 

```

Algorithm 1.2: The `PROPAGATE` function.

```

1: function PROPAGATE( $x$ )
2:    $\kappa \leftarrow \text{Ne}(x)$ 
3:   while  $k \notin \gamma$  do ▷ While the next random variable to encounter is not observed
4:      $v \sim p_\kappa(\text{Pa}(x))$  ▷ Sample the random variable
5:      $x \leftarrow x \cup \{(\kappa, v)\}$  ▷ Add the random variable to the state
6:      $\kappa \leftarrow \text{Ne}(x)$ 
7:   end while
8:   return  $x$ 
9: end function

```

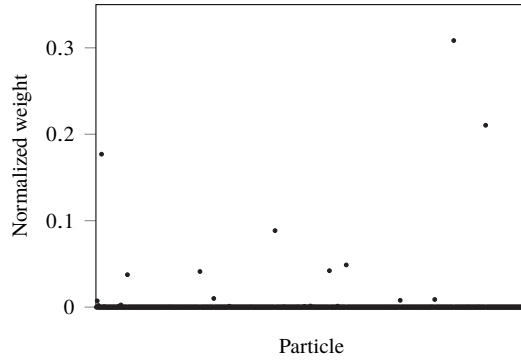


Figure 1.4: Demonstration of the weight degeneracy problem. See the description in the text.

Unfortunately, sequential importance sampling suffers from a problem known as *weight degeneracy* that limits its application: as t increases, the number of samples with extremely small normalized weights (the weights divided by their sum) increases. Eventually, a single sample will have the normalized weight approaching 1 and all remaining samples will have the normalized weights approaching 0. Figure 1.4 demonstrates the problem by showing the weights of 1000 samples for the linear Gaussian state-space model we used as an example on page 18 at time $t = 100$.

Note that this problem does not mean that sequential importance sampling is not useful, nor that it should not be used at all. For example, *inference compilation* [33], combining sequential importance sampling and neural networks, has been successfully used in probabilistic programming inference.

1.4.6 Sequential Monte Carlo

Sequential Monte Carlo (SMC) methods, also known as *particle filters*, addresses the weight degeneracy by resampling the samples, often denoted as *particles*, at certain times. The bootstrap particle filter (BPF), the simplest of the SMC methods, resamples the particles at each time step.

Extending the notation from the previous paragraph, let $\mathcal{S}_t = \{(x_t^n, w_t^n)\}_{n=1}^N$ denote the particle set at time t . The initial set \mathcal{S}_1 is constructed by sampling the state

$$x_1^n \sim p(x_1),$$

for each of the N particles, and calculating the corresponding weights

$$w_1^n = p(y_1 | x_1^n).$$

To propagate the particles from time $t - 1$ to time t , we construct the set $\mathcal{S}_t = \{(x_t^n, w_t^n)\}_{n=1}^N$ by following these steps for each particle:

1. **Resampling**—selecting one of the particles from \mathcal{S}_{t-1} with probabilities proportional to their weights (at time $t - 1$), identified by the *ancestor index* a_t^n :

$$a_t^n \sim \text{Categorical}(\{w_{t-1}^m\}_{m=1}^N).$$

For the sake of brevity in the notation, we allow the categorical distribution to be parameterized with unnormalized probabilities.

2. **Propagating** the selected particle to the next time step:

$$x_t^n \sim p(x_t | x_{t-1}^{a_t^n})$$

3. **Weighting** the propagated particle:

$$w_t^n = p(y_t | x_t^n)$$

Note that this procedure (probabilistically) discards particles with small weights, and that particles with high weights are likely to be selected and propagated to the next time step multiple times.

Once the final set \mathcal{S}_T has been constructed, we can use its particles and their weights to estimate the expected value of a test function of interest. To estimate the normalizing constant Z we need to use the weights from all time steps:

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

This estimator is unbiased (e.g. [12]), which allows us to use the BPF (and other SMC methods) in exact approximation methods (such as particle Markov Chain Monte Carlo).

Algorithm 1.3 shows how the bootstrap particle filter (BPF) is used for automatic inference in PPLs. The PROPAGATE function (Algorithm 1.2, p. 28) is the same as before. The biggest challenge when implementing this algorithm in probabilistic programming languages is related to handling of multiple concurrent executions: as we have mentioned above, a single particle might be chosen several times during resampling, which means that the inference engine needs to be able to copy, or clone, running executions. There are several strategies how to handle this efficiently, including using continuation-passing style (CPS) (e.g. [4]) or copy-on-write mechanisms (e.g. [45]).

Algorithm 1.3: Bootstrap particle filter (BPF) for probabilistic programming.

```

1: for  $n = 1, \dots, N$  do
2:    $w_0^n \leftarrow 1$ 
3:    $x_0^n \leftarrow \emptyset$  ▷ Initialization
4: end for
5: for  $t = 1, \dots, T$  do
6:   for  $n = 1, \dots, N$  do
7:      $a_t^n \sim \text{Categorical}(\{w_{t-1}^m\}_{m=1}^M)$  ▷ Resampling
8:      $x_t^n \leftarrow \text{PROPAGATE}(x_{t-1}^{a_t^n})$  ▷ Propagation
9:      $w_t^n \leftarrow p_{\gamma_t}(y_t | \text{Pa}(x_t^n))$  ▷ Weighting
10:     $x_t^n \leftarrow x_t^n \cup \{(\gamma_t, y_t)\}$ 
11:   end for
12: end for
13: for  $n = 1, \dots, N$  do
14:    $h^n \leftarrow h(x_T^n)$ 
15: end for
16:  $\mathbb{E}[h] \approx \sum_n w_T^n h^n / \sum_n w_T^n$ 

```

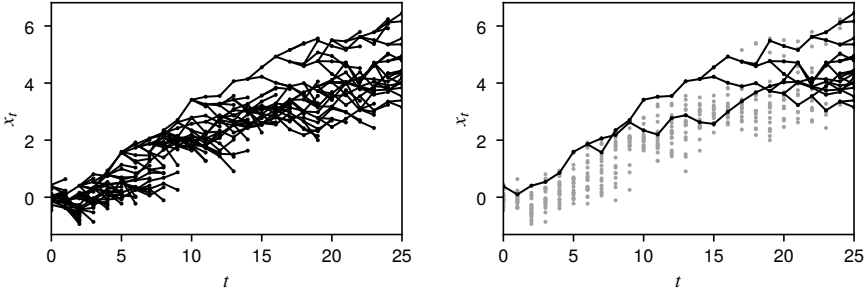


Figure 1.5: Path degeneracy problem. See the description in the text.

Although particle filters do not suffer from the weight degeneracy problem, they do suffer from a problem known as *path degeneracy* (e.g. [2]). Figure 1.5 illustrates the problem for a state-space model with one dimensional state. The plot on the left shows the states of all particles at all time steps. The lines visualize the ancestry history connecting the states of all particles $\{x_t^n\}$ at time t to the states of their ancestors $\{x_{t-1}^{a_t^n}\}$ at time $t - 1$. The plot on the right shows the complete ancestry paths of the particles at the very last time step. Note that for $t \leq 6$ all paths coalesce. In the context of probabilistic programming, this corresponds to a situation where all particles will have the same values of the random variables sampled early in the program.

To mitigate the path degeneracy problem, more advanced SMC methods do not

resample at each time step. Common practice is to use the effective sample size (ESS), given by

$$\text{ESS} = \frac{(\sum_n w_{t-1}^n)^2}{\sum_n (w_{t-1}^n)^2},$$

and resample only if the ESS drops below a chosen threshold (e.g. $N/2$). If the ESS is above the threshold, no resampling takes place and the ancestry indices are set deterministically: $a_t^n = n$. The problem might be further mitigated by using different methods of sampling the ancestry indices, including stratified and systematic resampling (e.g. [47]).

Despite the path degeneracy problem, the bootstrap particle filter and other SMC methods are well established as automatic inference engines used in PPLs. The propagation and weighting steps are embarrassingly parallelizable, and can run on separate CPUs, GPUs or cluster nodes.

Birth-death models of evolution

2.1 Introduction

In 1859 Charles Darwin in his influential work *On the Origin of Species* [11] introduced the idea that all species have descended from one “primordial form”, and that species evolve through natural selection and can split into separate lineages—a process known today as *speciation* (the term itself was coined first in 1906 by Orator F. Cook [9]). Together with *extinction*, when a species dies out, these ideas constitute the foundation of the theory of evolution, including phylogenetics, which focuses on the inference of the evolutionary history of species (and other taxonomic groups) and the relationships between them.

In this chapter we will introduce phylogenetic birth-death models, a family of continuous-time stochastic models of evolution in which the state, represented by all species living at a given time, can change in two possible ways:

- *speciation event* (representing birth), when one species splits into exactly two descendant species, and
- *extinction event* (representing death), when a species dies out.

The evolutionary process starts with a single species at some time $t_{\text{orig}} > 0$ in the past. A species undergoes a speciation after an exponentially distributed waiting time with the speciation rate λ , and goes extinct after an exponentially distributed waiting time with the extinction rate μ . These rates are inherited from the parent species at the speciation events (we will therefore also refer to these rates as *per-lineage* rates). In other words, starting at the origin, the speciation and extinction events follow a Poisson process with per-lineage rates λ and μ , and this process is “duplicated” at each speciation. Both descendants continue to follow their own independent processes. The speciation and extinc-

tion rates do not necessarily remain constant over time. In general, we expect that these rates may be subject to continuous or sudden changes that might affect a single species, but also a group of species or all species (e.g. in a mass extinction).

A couple of things require an explanation. First, we have used positive time for events in the past. Phylogeneticists often use *before present* (BP) time to denote how long ago something happened, for example, 8 *Ma ago* (*Mya* or *mya* are used as well) means 8 million years ago. There is a subtle difference between units of absolute time and relative time (i.e. duration), for example, 8 *Ma* (without “ago”), 8 *Myr* or 8 *myr* refer to a duration of 8 million years. Second, speciations are processes that take long time, rather than sudden events. However, considering the time scale of evolution, regarding speciations and extinctions as instantaneous events is a justified simplification.

2.2 Complete and surviving trees

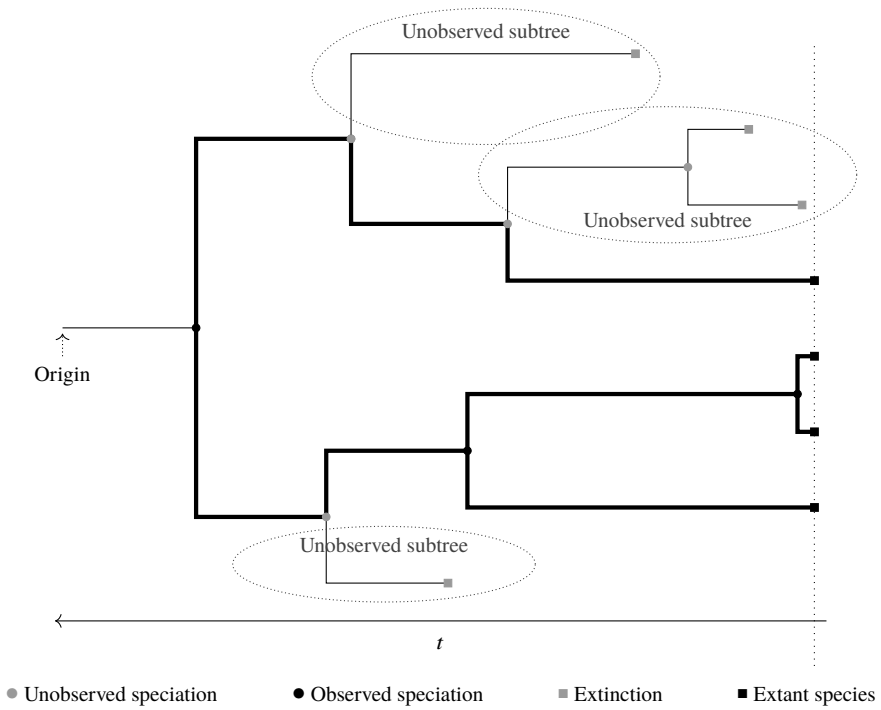
The result of a birth-death process (also called a diversification process) can be represented as a binary tree called a *phylogenetic tree*, or simply a *phylogeny*.

Its root represents the initial species at the time of origin t_{orig} ; internal nodes represent speciation events; and leaves represent extinction events (if in the past, $t > 0$) and currently living—*extant*—species (if in the present time, $t = 0$). The edge length is defined as the difference between the times of the connected nodes.

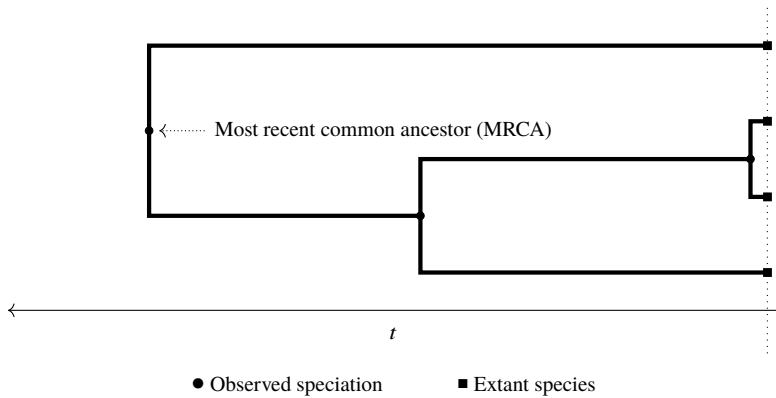
Figure 2.1a shows an example of a *complete* tree, representing the evolution of both the extant and extinct species. Note that the x -axis represents the time. Figure 2.1b depicts the corresponding *extant* tree, representing only the evolution of the extant species and their ancestors, with their *most recent common ancestor* (MRCA) at the root.

An example of a real surviving tree representing the evolution of cetaceans (whales, dolphins and porpoises) [57] is shown in Figure 2.2 (p. 36). Note that this tree is *labeled*—each extant species has a name. Surviving trees for taxonomic groups of interest are reconstructed from available data on the extant species, such as the genome sequences. Our goal in this thesis is to perform Bayesian inference of the model parameters given these reconstructed trees.

Let us introduce some necessary terminology: we will say that a speciation is *observed* if both descendants are ancestors of extant species, and *unobserved* or *hidden* otherwise. In Figure 2.1 we have used black circles to denote the observed speciations and gray circles to denote the unobserved ones.



(a) Complete tree.



(b) Surviving tree.

Figure 2.1: Example of a complete tree and its corresponding surviving tree.

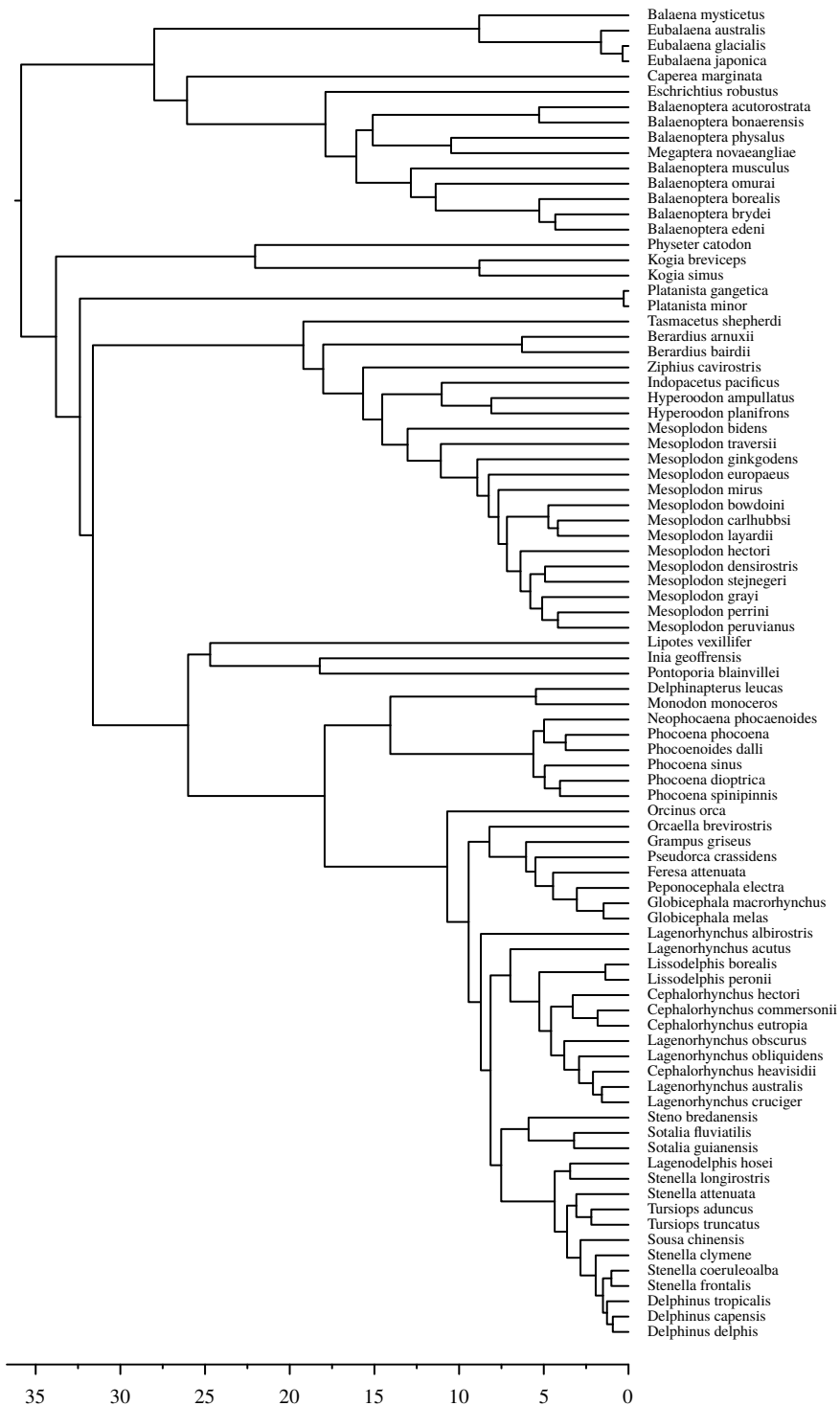


Figure 2.2: Extant phylogeny of cetaceans. Time axis in Mya.

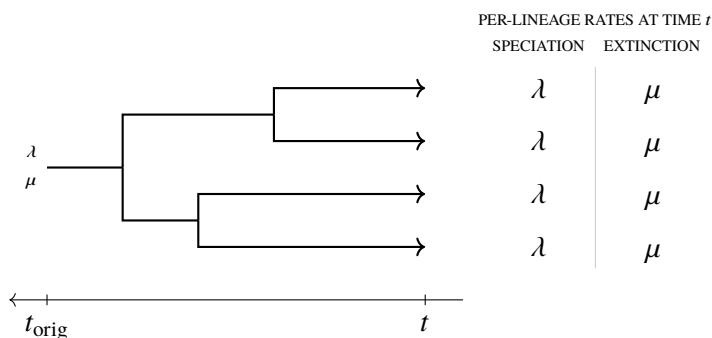


Figure 2.3: Per-lineage rates in the CRBD model.

Similarly, we will say that a subtree of a complete tree is *unobserved* or *hidden* if it does not include any extant species.

Given a complete tree, the corresponding surviving tree can be obtained by *pruning*: the nodes of the surviving tree are simply the observed speciations and the extant species; an edge (also called a branch) in the surviving tree means that there exists a path between the corresponding nodes in the complete tree that can only go through unobserved speciations. In Figure 2.1a (p. 35) we have marked these paths by thick lines.

2.3 Constant-rate birth-death model

The constant-rate birth-death (CRBD) model [16] is the simplest diversification model. As the name reveals, both the speciation rate λ and extinction rate μ are constant over time (Figure 2.3).

The generative probabilistic program for the CRBD model is shown in Algorithm 2.1 (p. 38). The `CRBD` function accepts the time of origin t_{orig} as well as the rates λ and μ , and returns a complete tree as a recursive data structure of the `NonEmptyTree` type. This type, defined at the beginning of the program (lines 1–2), uses the constructors `Extant`, `Extinction` and `Speciation`

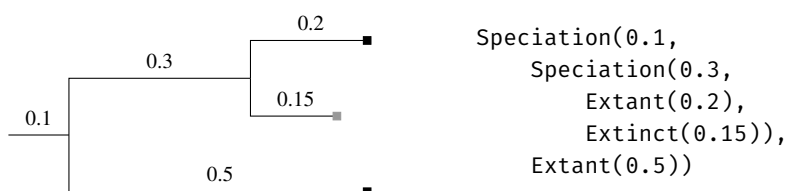


Figure 2.4: Example of a simple tree and its data representation.

Algorithm 2.1: Generative constant-rate birth-death model.

```
1: type NonEmptyTree = Extant(Real) | Extinction(Real) |
2:     Speciation(Real, NonEmptyTree, NonEmptyTree)
3: function CRBD( $t, \lambda, \mu$ )
4:      $\Delta \sim \text{Exponential}(\lambda + \mu)$ 
5:     if  $\Delta > t$  then
6:         return Extant( $t$ )
7:     end if
8:      $s \sim \text{Bernoulli}(\lambda/(\lambda + \mu))$ 
9:     if  $s$  then
10:        return Speciation( $\Delta$ , CRBD( $t - \Delta, \lambda, \mu$ ), CRBD( $t - \Delta, \lambda, \mu$ ))
11:    else
12:        return Extinction( $\Delta$ )
13:    end if
14: end function
```

representing different types of nodes. An example of a simple tree and its data representation is depicted in Figure 2.4 (p. 37). The first parameter of each constructor represent the length of the edge between the node and its parent; the second and third parameters of `Speciation` represent the subtrees. Also the complete tree in Figure 2.1a (p. 35) was generated using this algorithm with $t = 2$, $\lambda = 1$, and $\mu = 0.5$.

Let us go through the `CRBD` function in more detail: We draw a waiting time Δ until the next event (be it a speciation or an extinction) from an exponential distribution with rate $\lambda + \mu$ (line 4). If the waiting time is longer than the starting time t , the species has survived to the present time, and we return `Extant(t)` (line 6). Otherwise we decide if the event is a speciation (rather than an extinction) by a draw from the Bernoulli distribution with probability $\lambda/(\lambda + \mu)$ (line 8). If so, we call the `CRBD` function twice—once for each descendant—to generate the subtrees starting at time $t - \Delta$, and return a `Speciation` node using Δ and both subtrees as parameters (line 10). If not, we return `Extinct(Δ)` (line 12).

Recall the first chapter; it is the possibility of unbounded recursion in computationally universal probabilistic programming languages that allowed us to encode the generative `CRBD` model as a very simple probabilistic program.

We should mention here that there also exist pure birth models, that is, models with no extinction events. For example, the *constant-rate birth* (CRB) model, also known as the *Yule* model [64], is the simplest pure birth model with constant speciation rate. We will however consider such models as special cases of birth-death models with the extinction rate μ set to 0.

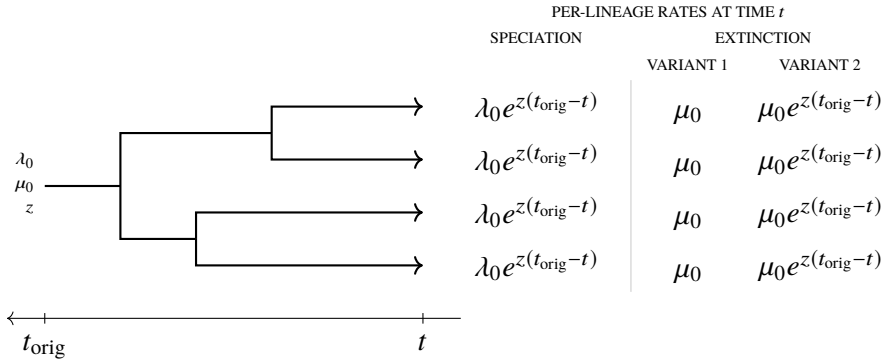


Figure 2.5: Per-lineage rates in the presented time-dependent birth-death model.

2.4 Selected non-constant-rate birth-death models

In this section we relax the assumption of the speciation and extinction rates being constant, and present a few models that we have been using in our experiments, and that allow continuous and/or sudden changes in the diversification rates.

2.4.1 Time-dependent birth-death models

Kendall [31] extended the CRBD model by allowing the speciation and extinction rates to be specified as functions of time. For example, consider the following function for the speciation rate:

$$\lambda(t) = \lambda_0 e^{z(t_{\text{orig}}-t)},$$

where λ_0 is the speciation rate at t_{orig} , and z is the trend parameter controlling how quickly the speciation rate might increase (when $z > 0$) or decrease (when $z < 0$). Choosing this particular form for the speciation rate is motivated by the empirical finding that diversification rates often slow down over time (e.g. [41]).

For the extinction rate we can consider two variants:

1. Keeping the extinction rate μ constant over time:

$$\mu(t) = \mu_0.$$

2. Keeping the turnover $\mu(t)/\lambda(t)$ constant over time:

$$\mu(t) = \mu_0 e^{z(t_{\text{orig}}-t)},$$

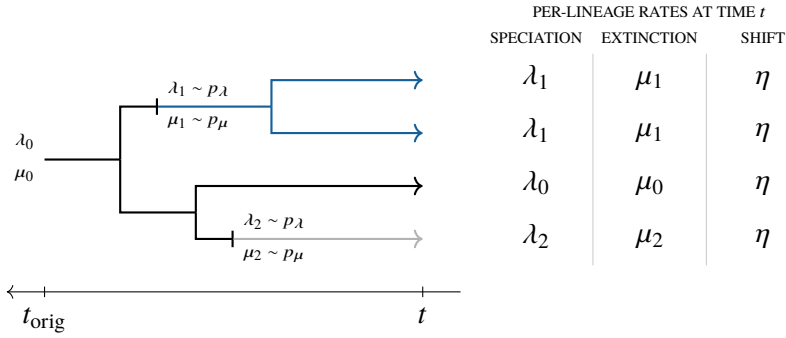


Figure 2.6: Per-lineage rates in the LSBDS model.

where μ_0 is the extinction rate at t_{orig} .

Figure 2.5 (p. 39) summarizes the rates for both variants.

2.4.2 Lineage-specific birth-death-shift model

The lineage-specific birth-death-shift (LSBDS) model by Höhna et al. [27] extends the CRBD model by introducing *diversification-rate shift* events. Like speciation and extinction events, also the shift events are related to a specific lineage (see Figure 2.6) and model sudden changes of the speciation and extinction rates.

Specifically, shift events follow a Poisson process with per-lineage rate η . Let λ_0 and μ_0 denote the initial speciation and extinction rate at the origin. At the i -th shift event the target species switches to a new process with the speciation and extinction rates drawn from the a priori chosen base distributions p_λ and p_μ :

$$\lambda_i \sim p_\lambda,$$

$$\mu_i \sim p_\mu.$$

At speciation events both descendants continue to follow the parent's process.

2.4.3 Bayesian analysis of macro-evolutionary mixtures

Bayesian analysis of macro-evolutionary mixtures (BAMM) by Rabosky [54] admits both exponentially increasing or decreasing speciation rates and rate shifts. The original model has been criticized in phylogenetic literature for its shortcomings by Moore et al. [42] and we follow their reinterpretation of the model.

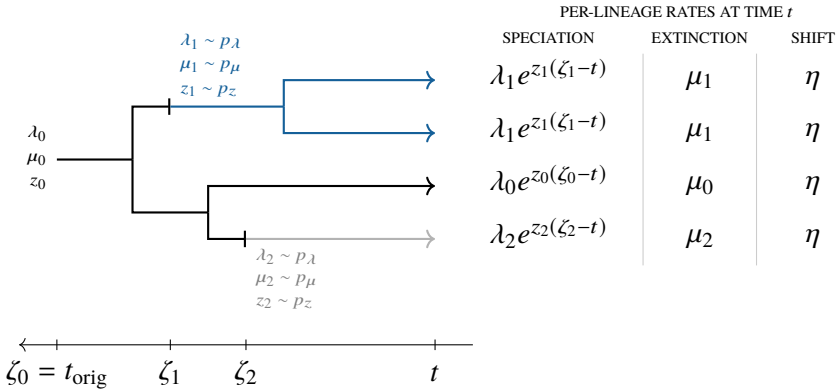


Figure 2.7: Per-lineage rates in the BAMM model.

Let λ_0 , μ_0 and z_0 denote the initial values of the speciation rate, the extinction rate and the trend parameter at the origin $\zeta_0 = t_{\text{orig}}$. The shift events follow a Poisson process with per-lineage rate η . At the i -th shift event, occurring at time ζ_i , the target species changes to a new process (identified by i) with the parameters λ_i , μ_i and z_i drawn from the a priori chosen base distributions p_λ , p_μ and p_z :

$$\begin{aligned}\lambda_i &\sim p_\lambda, \\ \mu_i &\sim p_\mu, \\ z_i &\sim p_z.\end{aligned}$$

The speciation and extinction rates for a species following the process i are given by

$$\begin{aligned}\lambda_i(t) &= \lambda_i e^{z_i(\zeta_i-t)}, \\ \mu_i(t) &= \mu_i.\end{aligned}$$

Figure 2.7 illustrates rate changes allowed by this model.

2.4.4 Cladogenetic diversification rate shift models

The cladogenetic diversification rate shift (ClaDS) models by Maliet et al. [36] are three models in which the speciation rate changes after each speciation event and remains constant until the next one. Each descendant draws its new speciation rate from a log-normal distribution:

$$\log \lambda \sim \mathcal{N}(\log(\alpha \lambda_p), \sigma^2),$$

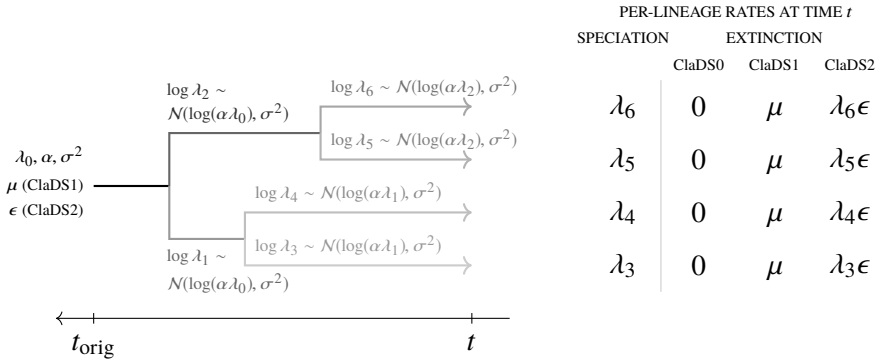


Figure 2.8: Per-lineage rates in the ClaDS models.

where λ_p is the speciation rate of the parent species, α and σ^2 are the model parameters modeling the trend and noise related to the speciation rate (Figure 2.8).

The three models differ in how they model the extinction rate:

- ClaDS0 does not allow any extinctions ($\mu = 0$),
- in ClaDS1 the speciation rate μ is constant for all species, and
- in ClaDS2 the turnover $\epsilon = \mu/\lambda$ is constant for all species.

2.4.5 Birth-death models with state

An important class of phylogenetic birth-death models are models with state. The main idea behind these is associating species with a state variable (for example to indicate if a species is herbivore or carnivore), and letting the speciation and extinction rates depend on this state.

In the binary state speciation and extinction model (BiSSE) by Maddison et al. [35] the speciation and extinction rates depend on a binary state $s \in \{0, 1\}$, but otherwise remain constant over time: λ_0 and μ_0 for $s = 0$, and λ_1 and μ_1 for $s = 1$. The state of a species is inherited at speciation events. The state changes at rate ζ_{01} when switching from state 0 to state 1, and ζ_{10} when switching from state 1 to state 0 (Figure 2.9). The states are typically assumed to be observed for some or all of the extant species.

The BiSSE model inspired a lot of similar models, such as MuSSE (multiple state speciation and extinction model) [18], QuaSSE (quantitative state speciation and extinction model) [17] and GeoSSE (geographic state speciation and extinction model) [22].

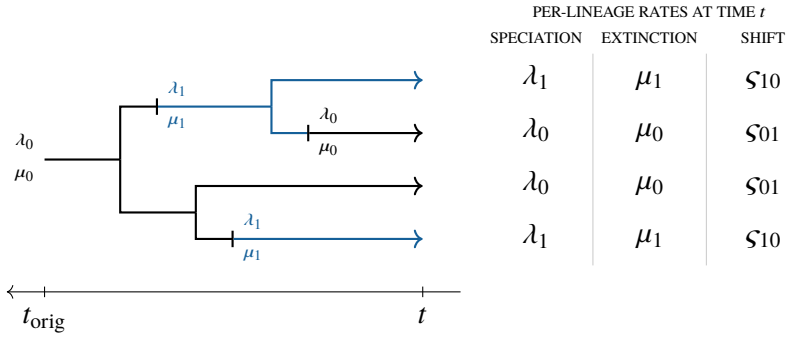


Figure 2.9: Per-lineage rates in the BiSSE model.

2.5 Bayesian inference of the model parameters

Let us now return to the inference problem. We are interested in inferring the parameters of the model based on the observed reconstructed (surviving) tree. As mentioned in the previous section, for the models with state, the observation might also be annotated with the state for a subset of the extant species.

Let θ denote the parameters of the model, and T the reconstructed tree. We assume that T is known without any error. Further, let $p(\theta)$ denote the prior distribution for the parameters θ . Our goal is to estimate the posterior distribution for the parameter set θ given the observed tree T :

$$p(\theta|T) = \frac{p(T|\theta)p(\theta)}{p(T)},$$

where $p(T|\theta)$ denotes the likelihood of the observed tree T given the parameters θ , and $p(T)$, the marginal likelihood (also called the model evidence), is given by

$$p(T) = \int p(T|\theta)p(\theta)d\theta.$$

One of the challenges we need to deal with is that, apart from a few of the simplest models, neither the posterior distribution nor the likelihood can be expressed in a closed form. In the next section we will show how to derive the likelihood for the CRBD model, which is one of these simple exceptions. Existing phylogenetic software, such as MrBayes [28], BEAST [15], BEAST 2 [6], RevBayes [29], RPANDA [43] or Diversitree [18], either implement only the models with analytical solutions or approximate the likelihood function e.g. by numerically solving differential equations describing the model, and use Markov chain Monte Carlo (MCMC) methods to sample from the posterior distribution.

In the next chapter, we will demonstrate how probabilistic programming with SMC based inference can be used to draw samples from the posterior distribution as well as to estimate the marginal likelihood $p(T)$ without bias. This is needed, for example, in analyses based on pseudo-marginal algorithms and for the model comparisons with Bayes factors.

2.6 Likelihood function for the CRBD model

As we have mentioned above, the constant-rate birth-death (CRBD) model is one of the few exceptions where we can express the likelihood of the observed tree in a closed form. In this section we derive the likelihood function, show the difference between the likelihood of an unlabelled oriented tree and the likelihood of a labelled unoriented tree, and conclude with a couple of possible approaches to dealing with an unknown time of origin.

Let us start with the extinction probability—the probability that an evolutionary process starting with a single species at time t goes extinct, or said differently, the probability that none of its ancestors survives to the present time.

For the sake of brevity, we will omit conditioning on $\theta = (\lambda, \mu)$ in the notation. Let $p_0(t)$ denote the extinction probability. This probability is a solution of the following differential equation:

$$\frac{dp_0(t)}{dt} = \mu + \lambda p_0(t)^2 - (\lambda + \mu)p_0(t),$$

with the boundary condition $p_0(0) = 0$.

To show that, let us consider what events can occur between $t + \Delta t$ and t , where $\Delta t \rightarrow 0$:

- an extinction event with probability $\mu\Delta t + O(\Delta t^2)$,
- a speciation event with probability $\lambda\Delta t + O(\Delta t^2)$, leading to two descendants, each going extinct with probability $p_0(t) + O(\Delta t)$,
- no events with probability $1 - (\lambda\Delta t + \mu\Delta t + O(\Delta t^2))$,
- more than one event with probability $O(\Delta t^2)$.

Putting everything together we get

$$\begin{aligned} p_0(t + \Delta t) &= \mu\Delta t \times 1 \\ &\quad + \lambda\Delta t \times p_0(t)^2 \\ &\quad + (1 - \lambda\Delta t - \mu\Delta t) \times p_0(t) \\ &\quad + O(\Delta t^2), \end{aligned}$$

which can be rewritten as

$$\frac{p_0(t + \Delta t) - p_0(t)}{\Delta t} = \mu + \lambda p_0(t)^2 - (\lambda + \mu)p_0(t) + O(\Delta t).$$

Taking the limit as $\Delta t \rightarrow 0$ leads to the above-mentioned differential equation. The boundary condition is due to zero probability of an extant species going extinct:

$$p_0(0) = 0.$$

The solution of this differential equation is given by

$$p_0(t) = 1 - \frac{\lambda - \mu}{\lambda - \mu e^{-(\lambda - \mu)t}}.$$

Now, let $p(t, t')$ denote the probability that a species living at time t survives until time t' , and that no observed speciation events occur during this time. This probability is a solution of the following differential equation:

$$\frac{\partial p(t, t')}{\partial t} = 2\lambda p_0(t)p(t, t') - (\lambda + \mu)p(t, t'),$$

with the boundary condition $p(t', t') = 1$.

To show that we will once again consider what events can occur between time $t + \Delta t$ and time t . The probability of no extinction events occurring in this interval is $1 - \mu\Delta t + O(\Delta t^2)$, and

- no speciation event occurs with probability $1 - \lambda\Delta t + O(\Delta t^2)$,
- one speciation event occurs with probability $\lambda\Delta t + O(\Delta t^2)$; in this case one of the *two* descendants must go extinct (probability of that being $p_0(t) + O(\Delta t)$) and the other must survive (probability of that being $p(t, t') + O(\Delta t)$),
- more than one speciation event occurs with probability $O(\Delta t^2)$.

Putting everything together we get

$$\begin{aligned} p(t + \Delta t, t') &= (1 - \mu\Delta t) \times (1 - \lambda\Delta t) \times p(t, t') \\ &\quad + 2 \times (1 - \mu\Delta t) \times \lambda\Delta t \times p_0(t) \times p(t, t') \\ &\quad + O(\Delta t^2), \end{aligned}$$

or, after rearranging the terms, the following equation:

$$\frac{p(t + \Delta t, t') - p(t, t')}{\Delta t} = 2\lambda p_0(t)p(t, t') - (\lambda + \mu)p(t, t').$$

Taking the limit as $\Delta t \rightarrow 0$ leads to the above-mentioned differential equation. At $t = t'$ the species has already survived to t' , i.e.

$$p(t', t') = 1.$$

The solution of this equation is given by

$$p(t, t') = \frac{e^{-(\lambda-\mu)t} \left(\lambda - \mu e^{-(\lambda-\mu)t'} \right)^2}{e^{-(\lambda-\mu)t'} \left(\lambda - \mu e^{-(\lambda-\mu)t} \right)^2}.$$

Let us now turn our attention to the surviving tree T . Let N denote the number of the extant nodes in T , and T' denote the tree obtained from T by adding the *stalk*—the edge from the origin to the most recent common ancestor (MRCA). To make the notation simpler, let us enumerate all nodes in T' by time (the order of the extant nodes is not important), using 0 for the origin at $t_0 = t_{\text{orig}}$, and $t_1 = t_{\text{MRCA}}, t_2, \dots, t_{N-1}$ denoting the times for the internal nodes. The time t_i for all extant nodes $i \in \{N, \dots, 2N - 1\}$ is equal to 0.

For the sake of simplicity we will assume that phylogenetic trees are unlabelled (i.e. the leaves have no labels) and oriented, that is, it is possible to distinguish which child of an internal node is the “left” one and which is the “right” one. This is indeed a simplification: the extant species have been assigned names and phylogenies are unoriented—if we swap the subtrees of any node, the new tree still represents the same result of the evolutionary process. The likelihood of a labelled and unoriented tree can be easily derived from the likelihood of the unlabelled and oriented tree by multiplying it with a constant factor that depends only on the number of extant nodes (e.g. [I]):

$$\tilde{p}(T|\theta) = \frac{2^{N-1}}{N!} p(T|\theta),$$

where $\tilde{p}(T|\theta)$ denotes the likelihood of the labelled and unoriented tree. As this constant factor does not affect the posterior distribution, such an assumption is quite common in the phylogenetic literature, often even without being mentioned explicitly.

The likelihood $p(T'|\theta)$ of the extended tree T' is given by the product of the following factors:

- for each edge in T' from a parent node i to a child node j , the probability of the species surviving along the edge and not containing an observed speciation, given by

$$p(t_i, t_j) = \frac{e^{-(\lambda-\mu)t_i} \left(\lambda - \mu e^{-(\lambda-\mu)t_j} \right)^2}{e^{-(\lambda-\mu)t_j} \left(\lambda - \mu e^{-(\lambda-\mu)t_i} \right)^2},$$

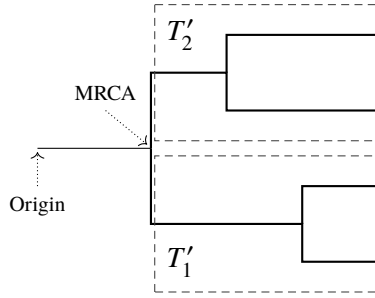


Figure 2.10: Descendants of the MRCA.

- for each internal node $i \in \{1, \dots, N - 1\}$, the probability of a speciation event at the end of the edge from the parent of i to i , given by λ .

Putting everything together we can express the likelihood of the extended surviving tree T' as follows:

$$p(T'|\theta) = \lambda^{N-1}(\lambda - \mu)^{2N} \prod_{i=0}^{N-1} \frac{e^{-(\lambda-\mu)t_i}}{(\lambda - \mu e^{-(\lambda-\mu)t_i})^2}.$$

We have in the derivation of $p(T'|\theta)$ used a tree extended with the stalk, the edge from the origin to the MRCA, but we do not actually know t_{orig} . One option is to define the likelihood of T as follows:

$$p(T|\theta) = p(T'_1|\theta)p(T'_2|\theta),$$

where T'_1 and T'_2 denote the subtrees of T corresponding to the two descendants of the MRCA (the apostrophes are used to emphasize that both these subtrees have a stalk), see Figure 2.10.

A better option is to condition the posterior distribution on t_{MRCA} as well (e.g. [56]):

$$p(\theta|T, t_{\text{MRCA}}) = \frac{p(T|\theta, t_{\text{MRCA}})p(\theta)}{p(T|t_{\text{MRCA}})},$$

where the likelihood is given by

$$p(T|\theta, t_{\text{MRCA}}) = \frac{p(T'_1|\theta)}{S(\theta, t_{\text{MRCA}})} \frac{p(T'_2|\theta)}{S(\theta, t_{\text{MRCA}})}.$$

$S(\theta, t)$ denotes the *survival probability*—the probability that an evolutionary process starting at time t with a single species produces at least one extant species. The survival is complementary to the extinction, i.e. $S(\theta, t) + p_0(t) = 1$.

The model evidence $p(T|t_{\text{MRCA}})$ is in this case given by

$$p(T|t_{\text{MRCA}}) = \int p(T|\theta, t_{\text{MRCA}})p(\theta)d\theta.$$

Probabilistic programming for phylogenetics

3.1 Parameter inference for the CRBD model

Recall the recipe for probabilistic modeling with probabilistic programming languages from the first chapter:

1. Write a generative model as a computer program.
2. Add observe statements.
3. Run the program to automatically do the inference.

For phylogenetic birth-death models, the challenge is related to the second point. Where and how exactly should we specify the observed tree? Can we just implement the generative CRBD model (Algorithm 2.1, p. 38) from the previous chapter, implement a function to prune a complete tree (to return its surviving tree), and condition on the surviving tree being equal to the observed tree (in the sense that it has the same topology and that all edge lengths are the same)? Unfortunately, the answer is no (at least at the time of writing this thesis). To understand why, consider the following example:

```
x ~ N(0, 1)
... do something with x ...
observe x = 0.1
```

Since the variable x is continuous, the probability of x being equal to 0.1 (or any other value) is 0. This is indeed the reason why we need both the distribution and the observed value to be specified for continuous variables. Fortunately, we can fix this program rather easily:

```
x ← 0.1
observe x ~ N(0, 1)
... do something with x ...
```

Can we fix the following program in a similar way?

```
x ~ N(0, 1)
... do something with x ...
observe f(x) = 0.1
```

It depends on $f(x)$. For example, if it is an injective and differentiable function, we can use the change of variables technique to determine the distribution of $f(x)$. However, for more complex functions, including pruning, we need to choose a different method.

Our approach [II] is based on augmentation, i.e. obtaining a complete tree from the observed one. We traverse the observed tree, and for each edge we use the generative model to simulate all unobserved events: hidden speciations, parameter shifts, state shifts, etc. For each hidden speciation, we use the model also to simulate the evolution of extinct species (starting at the time of the hidden speciation).

As we traverse the observed tree, and augment it with unobserved events and subtrees, we condition on the following:

1. No extinction events occur along the observed edges.
2. Observed speciation events occur at the end of the corresponding edges.
3. No species survive to the present time during simulations of unobserved subtrees. This implies setting the importance weight to 0 (and stopping the execution) if any simulated species survives to the present time (otherwise the species would be observed and had to be a part of the observed tree).
4. We double the probability of the execution for each hidden speciation event. This is related to the fact that either of two descendants can correspond to the hidden subtree.

A probabilistic program for the CRBD model in pseudocode is shown in Algorithm 3.1. The program uses a specific probabilistic construct called **factor** to explicitly multiply the probability of the execution (i.e. the particle weight in the SMC based inference methods) by the given value. The factor construct is closely related to the observe construct, which multiplies the probability by the likelihood of the observed value with respect to the given distribution and

Algorithm 3.1: A probabilistic program for the CRBD model.

```
1:  $\lambda \sim$  prior distribution for  $\lambda$ 
2:  $\mu \sim$  prior distribution for  $\mu$ 
3: for all  $r \in$  nodes of  $T$  do
4:   if  $r$  is the root then
5:     continue
6:   end if
7:    $c_{\text{hs}} \sim \text{Poisson}(\lambda\Delta_r)$ 
8:   for  $i \leftarrow 1$  to  $c_{\text{hs}}$  do
9:      $t \sim \text{Uniform}(t_r, t_r + \Delta_r)$ 
10:    if  $\text{SURVIVES}(t, \lambda, \mu)$  then
11:      factor 0
12:    end if
13:    factor 2
14:  end for
15:  observe 0  $\sim \text{Poisson}(\mu\Delta_r)$ 
16:  if  $r$  is a speciation then
17:    observe 0  $\sim \text{Exponential}(\lambda)$ 
18:  end if
19: end for
20: return  $\lambda, \mu$ 

21: function  $\text{SURVIVES}(t, \lambda, \mu)$ 
22:    $\Delta \sim \text{Exponential}(\mu)$ 
23:   if  $\Delta \geq t$  then
24:     return true
25:   end if
26:    $c_b \sim \text{Poisson}(\lambda\Delta)$ 
27:   for  $i \leftarrow 1$  to  $c_b$  do
28:      $t' \sim \text{Uniform}(t - \Delta, t)$ 
29:     if  $\text{SURVIVES}(t', \lambda, \mu)$  then
30:       return true
31:     end if
32:   end for
33:   return false
34: end function
```

its parameters. Note that we have not implemented conditioning on the time of the MRCA, we will return to this in Section 3.4 (p. 63).

Let us go through the program in more detail. After defining the rates (lines 1–2), we traverse all nodes of the observed tree T in a loop (line 3), for example in the depth-first manner. We skip the root node (since the stalk is not known) (line 4). The current node is denoted by r , the corresponding time by t_r , and the length of the edge to its parent by Δ_r . To sample the times of hidden speciation events along this edge we could sample a waiting time from an exponential distribution with rate λ , reduce the current time (initially $t_r + \Delta_r$) by the sampled time, and repeat this procedure until the current time falls below t_r (recall that we use the time before present).

We use an alternative approach: we sample the number of hidden speciation events c_{hs} along the edge from a Poisson distribution with rate $\lambda\Delta_r$ (line 7). Given the number of events, their times are distributed uniformly (line 9). For each hidden speciation we call the `SURVIVES` function (line 10). This function essentially implements the generative CRBD model to simulate the evolution of extinct species. It interrupts the simulation and returns true immediately if any species survives to the present time. In that case the execution probability is set to 0 (line 11). Note that the inference engine will terminate the execution at this point. If the simulation only led to extinct species (i.e. the `SURVIVES` function returned false), we double the execution probability (line 13) and continue to the next hidden speciation.

Finally, we observe that there were no extinction events along the current edge (line 15). More specifically, we use a Poisson distributed random variable with rate $\mu\Delta_r$ to represent the number of extinction events, and observe that this number is 0.

If the current node represents a speciation event (line 17), we use an exponentially distributed random variable with rate λ , representing the waiting time for the next speciation event (starting at time t_r , since the speciation events between $t_r + \Delta_r$ and t_r have been simulated earlier), and observe that this time is 0.

In the `SURVIVES` function, we first sample a waiting time Δ until the next extinction (line 22). If it is greater than the starting time t , the species has survived to the present time and we return true. Otherwise we sample the number of speciation events along a branch of length Δ (line 26). For each of them we sample the speciation time from a uniform distribution (line 28) and recursively call the `SURVIVES` function to simulate the evolution starting at that time.

With the exception of the CRBD model we need to iterate over the nodes in

an order which ensures that a node is processed before its descendants, due to the inheritance of the rates (and possibly the state). During our work on [I] we experimented with different orderings: we first generated a few random “permutations” for each observed tree by swapping the left and right subtrees of each node with probability 1/2. For each permutation we ran the program multiple times, using depth-first search, and collected estimates of the marginal likelihood. The variances of the estimates for different permutations of the same observed tree varied significantly, indicating that the traversal order had a significant effect on the quality of the inference. We also discovered that processing the subtree with the smaller total length (i.e. the sum of all edge lengths) first is an easy and effective heuristic to keep the variance low. In the future we plan to look at this question in more detail and try to find other heuristics or methods to determine the optimal traversal order.

In Section 1.4 (p. 24) on SMC based inference we resampled at each observe. Some programming languages (e.g. WebPPL) will do so also at each factor. Due to a random number of hidden speciations, this can create a misalignment of which part of the tree is currently processed by the particles, and lead to poor performance of the particle filter. In our implementation we resample the particles after processing each node of the observed tree. Details and comparison of unaligned and aligned resampling can be found in [I].

To estimate the posterior distribution of the rates λ and μ , we just need to collect the sampled values from all particles (line 20) and their weights. As the program sampled a complete tree during the execution, we can easily change the return statement in order to estimate the expected value of any test function of the rates and/or the complete tree.

The normalization constant Z coincides with the likelihood $p(T|\theta)$. This is indeed one of the very important advantages of our approach, since it enables comparison of different models using Bayes factors, as we have shown in [I]. The estimate of the normalization constant is unbiased, which makes it possible to use it in hierarchical inference, such as particle Markov chain Monte Carlo (PMCMC) [3]. If we also include the factor $2^{N-1}/N!$ (e.g. at the very beginning of the program), the normalization constant will represent the likelihood $\tilde{p}(T|\theta)$ of the labelled and unoriented tree.

Until now we have assumed ideally reconstructed trees that include all extant species. This is not always the case in real life. In birth-death models, *incomplete sampling* of extant species (when some of the extant species and their ancestors are missing in the reconstructed tree) can be easily modeled in several ways, including *uniform sampling* (e.g. [55]): each species at the present time $t = 0$ is included in the reconstructed tree with probability ρ . Algorithm 3.2 (p. 55) shows how to extend the program to support this model of incomplete

sampling. The parts that are different from Algorithm 3.1 are shown in blue.

All ideas and methods we have introduced in this section and demonstrated for the CRBD model are applicable to any birth-death model. Of course, the programs need to handle other types of events as well, implement the formulas calculating the speciation and extinction rates, pass all necessary information from a parent node to its descendants, etc.

One of the non-trivial implementation details is sampling of the event times of a non-homogenous Poisson process. Let $\nu(t)$ denote the rate of the process and Δ its duration. If there exists ν_0 such that $\nu_0 \geq \nu(t)$ for all $t \in (0, \Delta)$, we can use the *thinning* algorithm [34]:

```
t ← 0
while t < Δ do
  δ ~ Exponential(ν0)
  t ← t + δ
  if t < Δ then
    α ~ Bernoulli(ν(t)/ν0)
    if α then
      yield t
    end if
  end if
end while
```

For decreasing $\nu(t)$ it is convenient to set ν_0 at the beginning of each loop iteration to $\nu(t)$.

Our implementation of the CRBD model (and other models from Chapter 2) in Birch can be found at <https://github.com/phypppl/probabilistic-programming>.

3.2 Alive particle filter

While simulating the hidden subtrees, if any of the species survives to the present time, we set the weight to zero and terminate the execution. These executions impoverish the particle set: there are fewer particles (with non-zero weight) to choose from at the next resampling checkpoint. But is this a problem at all? How often does a simulation of a hidden subtree result in setting the weight to zero? For the CRBD model, this probability is given by $1 - p_0(t)$, where $p_0(t)$ is the extinction probability introduced in Section 2.6 (p. 44):

$$1 - p_0(t) = \frac{\lambda - \mu}{\lambda - \mu e^{-(\lambda - \mu)t}}.$$

Algorithm 3.2: A probabilistic program for the CRBD model with uniform incomplete sampling.

```

1:  $\lambda \sim$  prior distribution for  $\lambda$ 
2:  $\mu \sim$  prior distribution for  $\mu$ 
3: for all  $r \in$  nodes of  $T$  do
4:   if  $r$  is the root then continue end if
5:    $c_{\text{hs}} \sim \text{Poisson}(\lambda\Delta_r)$ 
6:   for  $i \leftarrow 1$  to  $c_{\text{hs}}$  do
7:      $t \sim \text{Uniform}(t_r, t_r + \Delta_r)$ 
8:     if  $\text{SURVIVES}(t, \lambda, \mu)$  then
9:       factor 0
10:    end if
11:    factor 2
12:  end for
13:  observe  $0 \sim \text{Poisson}(\mu\Delta_r)$ 
14:  if  $r$  is a speciation then
15:    observe  $0 \sim \text{Exponential}(\lambda)$ 
16:  end if
17:  if  $r$  is an extant species then
18:    observe  $\text{true} \sim \text{Bernoulli}(\rho)$ 
19:  end if
20: end for
21: return  $\lambda, \mu$ 
22: function  $\text{SURVIVES}(t, \lambda, \mu)$ 
23:    $\Delta \sim \text{Exponential}(\mu)$ 
24:   if  $\Delta \geq t$  then
25:      $o \sim \text{Bernoulli}(\rho)$ 
26:     if  $o$  then
27:       return true
28:     end if
29:      $\Delta \leftarrow t$ 
30:   end if
31:    $c_b \sim \text{Poisson}(\lambda\Delta)$ 
32:   for  $i \leftarrow 1$  to  $c_b$  do
33:      $t' \sim \text{Uniform}(t - \Delta, t)$ 
34:     if  $\text{SURVIVES}(t', \lambda, \mu)$  then return true end if
35:   end for
36:   return true
37: end function

```

Algorithm 3.3: Alive particle filter (APF) for probabilistic programming.

```

1: for  $n = 1, \dots, N$  do
2:    $w_0^n \leftarrow 1$ 
3:    $x_0^n \leftarrow \emptyset$  ▷ Initialization
4: end for
5: for  $t = 1, \dots, T$  do
6:    $P_t \leftarrow 0$ 
7:   for  $n = 1, \dots, N + 1$  do
8:     repeat
9:        $a_t^n \sim \text{Categorical}(\{w_{t-1}^m\}_{m=1}^M)$  ▷ Resampling
10:       $x_t^n \leftarrow \text{PROPAGATE}(x_{t-1}^{a_t^n})$  ▷ Propagation
11:       $P_t \leftarrow P_t + 1$ 
12:       $w_t^n \leftarrow p_{\gamma[t]}(y_t | \text{Pa}(x_t^n))$  ▷ Weighting
13:       $x_t^n \leftarrow x_t^n \cup \{(\gamma[t], y_t)\}$ 
14:     until  $w_t^n > 0$ 
15:   end for
16: end for
17: for  $n = 1, \dots, N$  do
18:    $h^n \leftarrow h(x_T^n)$ 
19: end for
20:  $\mathbb{E}[h] \approx \sum_n w_T^n h^n / \sum_n w^n$ 

```

For most models, this probability does not have a closed form (that is the reason why we use the generative model to simulate the hidden subtrees). Taken into account that there might be several hidden speciations along a single edge, the fraction of particles that have zero weight at a resampling checkpoint might be quite large.

In [II] we proposed a solution based on an *extended alive particle filter* (APF). This inference algorithm replaces every particle that has zero weight by returning to the previous time step and repeating the resampling and propagation steps. The procedure is repeated until all particles have positive (i.e. non-zero) weight. The APF needs to use one extra particle (compared to the bootstrap particle filter), but with a reasonable number of particles this cost is negligible. Algorithm 3.3 summarizes the algorithm (differences between this algorithm and the bootstrap particle filter are shown in blue).

The estimator of the marginal likelihood is given by

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

where P_t is the total number of the propagations at time step t , including the propagations for rejected particles and the propagations for the extra particle.

More details, including the proof of the unbiasedness of the estimator of the marginal likelihood can be found in [II]. There we also present experiments comparing the performance of the alive particle algorithm with the bootstrap particle filter for the CRBD and BiSSE models.

In [IV] we present a similar algorithm, *particle filter with rejection control* (PF-RC), in which the weights of the particles are compared to given thresholds (rather than to 0 as in the APF). Particles with weights below the chosen thresholds, are probabilistically discarded (with probabilities given by the weights of the particles divided by the threshold) and the resampling and propagation steps are repeated. We have demonstrated the algorithm with a couple of examples not related to phylogenetics, but believe that this algorithm might be useful for the phylogenetic birth-death models too. The question that needs to be investigated is how to determine the thresholds. An interesting incentive for using this algorithm for phylogenetic models is that it enables the rejection of some particles even before simulating the hidden subtrees. For each node, we can simulate the number of hidden speciation events and update the weight, assuming that no species survive while simulating the hidden subtrees, and reject the particles based on the updated weights. For particles that are (tentatively) accepted, we can simulate the hidden subtrees, and reject those where the assumption is not met (i.e. any species survives to the present time).

3.3 Delayed sampling

Recall the following example in Birch from the first chapter:

```
x:Random<Real>;
y:Random<Real>;

x ~ Gaussian(0, 1);
y ~ Gaussian(x, 1);
stdout.print("y = " + y.value() + "\n");
```

As we mentioned there, the variable x is never realized during the execution of the program. From a mathematical point of view, this makes perfect sense. The program specifies the joint distribution $p(x, y)$, and if sampled immediately, we would first sample x from $\mathcal{N}(0, 1)$, and then y , based on the value of x , from $p(y|x)$, which is $\mathcal{N}(x, 1)$. Due to the conjugacy relationship between x and y there is a closed form solution for both $p(y)$ and $p(x|y)$:

$$p(y) = \int_{-\infty}^{\infty} p(y|x)p(x)dx = \mathcal{N}(y|0, 2)$$

$$p(x|y) = \mathcal{N}(x|y/2, 1/2)$$

We can sample y from $\mathcal{N}(0, 2)$ first, and then sample x from $\mathcal{N}(y/2, 1/2)$. We are indeed getting samples from the same joint distribution encoded by the program, since $p(x, y) = p(x)p(y|x) = p(y)p(x|y)$.

Delayed sampling, introduced in [III], exploits analytically tractable relationships between random variables in probabilistic programs automatically in order to lower the variance of the resulting estimators. Programmers do not need to implement any of the analytical relationships (or even realize that there actually are some).

When executing a program, a directed acyclic graph, or more precisely a forest of trees, is built dynamically, with the nodes representing the encountered random variables and the edges representing the analytical relationships between them. Each node and the corresponding variable can be in one of the three states:

- *initialized* (the random variable has been inserted into the graph but not processed further),
- *marginalized* (the random variable has been marginalized over its parents and potentially conditioned on the value of the dependent variable),
- *realized* (the random variable has been sampled or it is observed).

Nodes in the marginalized state have a proposal distribution associated with them, which is used when realizing the random variables. The method does not allow marginalized nodes to have more than one marginalized child. This means that the marginalized nodes in each tree form a path, which we will refer to as the *M-path*. The M-path of a tree starts at the root node (which is always in the marginalized state) and ends in a node referred to as the *terminal* node.

Each random variable encountered during the execution is inserted into the graph. If there is an analytically tractable relationship to another variable, it is inserted as a child of the node corresponding to that variable and set to be in the initialized state, otherwise it forms a new tree consisting of a single node in the marginalized state (its proposal distribution is the distribution specified by the program).

For a random variable to be realized, the corresponding node must be the terminal node on an M-path. If it already is, the value is sampled from (resp. observed with respect to) its proposal distribution, and the state is changed to realized. The edges to the children are removed, and each child becomes the marginalized root of a new tree (the proposal distribution is given by the

program and the sampled value). The parent of the realized node (if there is one) becomes the new terminal node of the M-path and its proposal distribution is conditioned on the realized value.

If the target node corresponding to the random variable being realized is not a terminal node, the following graph operations must be performed before we can use the above-mentioned procedure:

- If the node lies on the M-path: the M-path is shortened by repeatedly realizing its terminal node until the target node becomes terminal (this operation is called *pruning*).
- If the node does not lie on the M-path: we first use pruning to shorten the M-path until any of the target node's ancestors becomes terminal, and then extend the M-path towards the target node by repeatedly marginalizing the initialized variables over their parents (this operation is called *grafting*).

Specific details of these operations, including the pseudocode, as well as a comparison of the performance of immediate and delayed sampling can be found in [III].

To illustrate these operations, consider the following program:

```

a ~ N(0, 1)
b ~ N(a, 1)
c ~ N(b, 1)
d ~ N(b, 1)
e ~ N(c, 1)
f ~ N(d, 1)
PRINT(e)
PRINT(d)

```

Figure 3.1 (p. 60) shows different states of the graph during execution. Figure (a) corresponds to the state before printing the value of e .

When the value of e needs to be sampled, we first marginalize b , then c and finally e , in order to form the M-path from the root a to the node e . The state of the graph at this moment is depicted in Figure (b) together with the proposal distributions.

When e gets sampled from q_e , the sampled value, say 1.2, is used to update the proposal distribution q_c . The state of e is changed to realized, and the edge between c and e is removed. The node c becomes the new terminal node. This state corresponds to Figure (c).

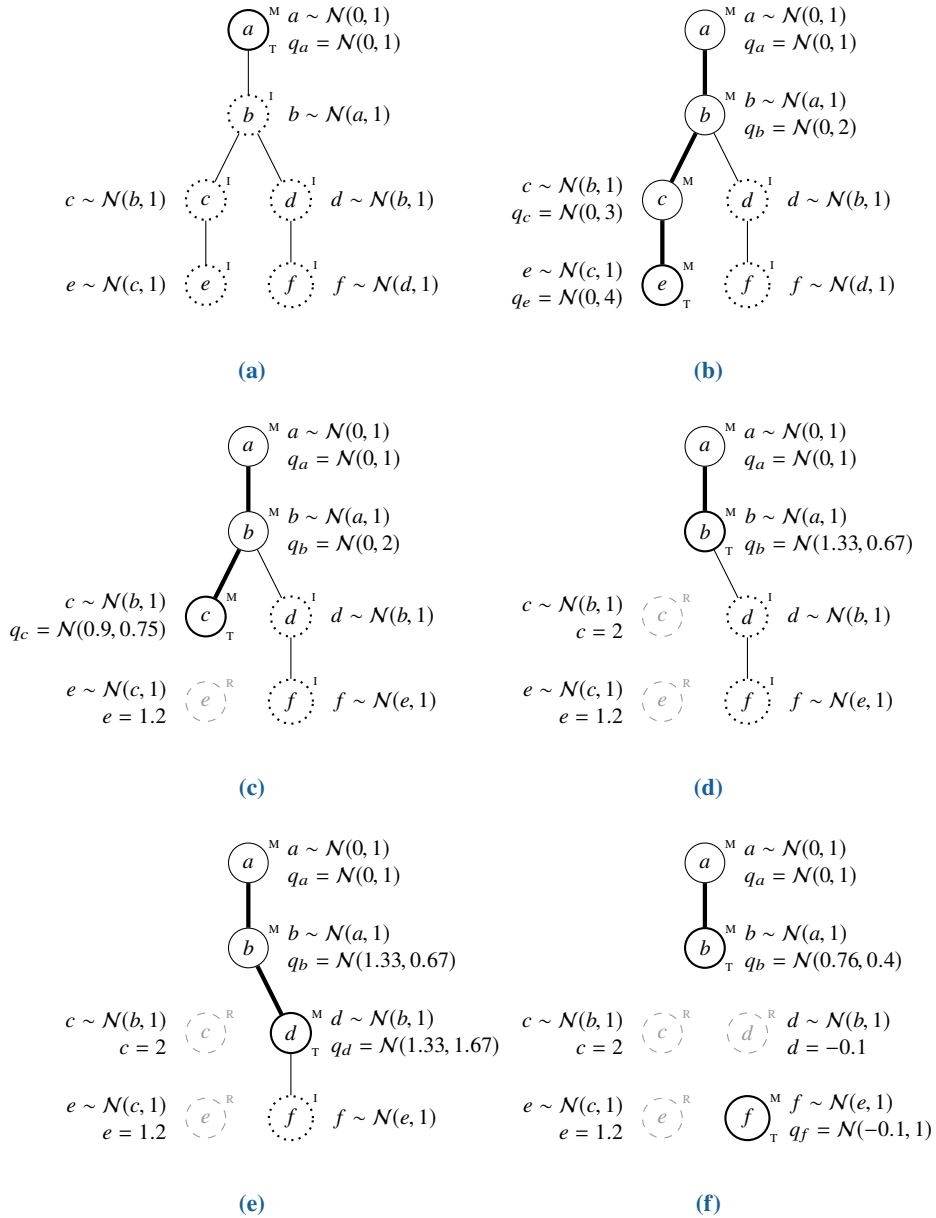


Figure 3.1: Illustration of graph operations in delayed sampling. See the description in the text. Initialized nodes are shown with dotted circles and the letter I, marginalized nodes with solid circles and the letter M, and realized nodes with dashed gray circles and the letter R. The letter T denotes terminal nodes on M-paths, which are shown with thick lines.

Next we wish to print the value of d . Before we can sample it, we need d to become a terminal node. We first shorten the M-path by sampling c (and the sampled value, say 2, is used to update the proposal distribution q_b), see Figure (d), and then extend it to d by marginalization of this variable, see Figure (e). At this moment we can sample the value of d , and use the value, say -0.1 , to update the proposal distribution of b . The node for f becomes the root of a new tree, and the proposal distribution is set based on the sampled value.

Let us now return to the phylogenetic birth-death models, and show how delayed sampling can be employed in these models [II]. It is mathematically convenient to use gamma distributions as priors for both λ and μ , since the gamma distribution is a conjugate prior for both the exponential and Poisson likelihood:

$$\begin{aligned}\lambda &\sim \text{Gamma}(k_\lambda, \theta_\lambda), \\ \mu &\sim \text{Gamma}(k_\mu, \theta_\mu).\end{aligned}$$

Note that we use the shape/scale parametrization for gamma distributions.

Let us go through the probabilistic constructs involving the speciation and extinction rates in Algorithm 3.1 (p. 51) and Algorithm 3.2 (p. 55):

1. Sampling the number of speciation events c along an edge of length Δ :

$$c \sim \text{Poisson}(\lambda\Delta).$$

In the delayed sampling setting, if λ is marginalized and its proposal distribution is

$$q_\lambda = \text{Gamma}(k, \theta),$$

the value of c is sampled from the marginal distribution

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

where the sampled value represents the number of failures. (The parameterization used here is the number of successes before the experiment is stopped, and the success probability.)

The proposal distribution for λ is then updated, based on the sampled value c , to

$$q_\lambda \leftarrow \text{Gamma}\left(k + c, \frac{\theta}{1 + \Delta\theta}\right),$$

that is, its shape gets incremented by the sampled value, and the scale is updated to $\theta/(1 + \Delta\theta)$.

2. Observing no extinction events along an edge of length Δ :

$$\text{observe } 0 \sim \text{Poisson}(\mu\Delta)$$

Similarly, if μ is marginalized with the proposal distribution being

$$q_\mu = \text{Gamma}(k, \theta),$$

the importance weight is multiplied by the probability mass of 0 with respect to the proposal distribution q_c of a random variable representing the number of extinction events along the edge with length Δ :

$$q_c = \text{NegativeBinomial}\left(k, \frac{1}{1 + \Delta\theta}\right).$$

The proposal distribution for μ is then updated to

$$q_\lambda \leftarrow \text{Gamma}\left(k, \frac{\theta}{1 + \Delta\theta}\right),$$

that is, its shape remains the same, and the scale gets updated to $\theta/(1 + \Delta\theta)$.

3. Sampling a waiting time Δ from an exponential distribution with rate μ :

$$\Delta \sim \text{Exponential}(\mu).$$

Again, we assume that the proposal distribution for μ is

$$q_\mu = \text{Gamma}(k, \theta),$$

and sample the value of Δ from

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

where we have used the scale/shape parametrization of the Lomax distribution (see Appendix A, p. 79).

The proposal distribution for μ is afterwards updated to

$$q_\mu \leftarrow \text{Gamma}\left(k + 1, \frac{\theta}{1 + \Delta\theta}\right),$$

that is, its shape gets incremented by 1, and the scale gets updated to $\theta/(1 + \Delta\theta)$.

4. Finally, observing the speciation event at the end of an observed edge:

observe $0 \sim \text{Exponential}(\lambda)$.

Starting with the proposal distribution of λ ,

$$q_\lambda = \text{Gamma}(k, \theta),$$

the importance weight is multiplied by the probability density of 0 with respect to the proposal distribution q_Δ of a random variable representing the waiting time:

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

The proposal distribution of λ gets updated to

$$q_\lambda = \text{Gamma}(k + 1, \theta),$$

that is, its shape gets incremented by 1, and the scale remains the same.

All four situations are summarized in Figure 3.2 (p. 64).

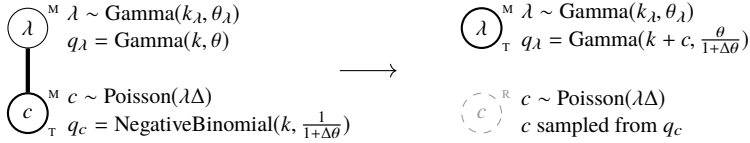
Interestingly, the consequence of employing delayed sampling in Algorithm 3.1 (p. 51) and Algorithm 3.2 (p. 55) is that both λ and μ remain delayed even when the program finishes. This allows us to estimate the posterior distribution of λ resp. μ as a mixture of the particles' proposal distributions at the end of the execution. The mixture weights are just the normalized final weights of the particles.

More details about using delayed sampling with the alive particle filter (APF) for phylogenetic birth-death models, including a comparison to immediate sampling with the bootstrap particle filter (BPF) can be found in [II]. The combination of APF with delayed sampling has also been used to run the inference in Birch in the experiments in [I].

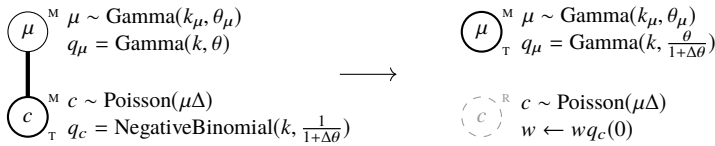
3.4 Conditioning on the time of the most recent common ancestor

At the end of Chapter 2 we considered two options for dealing with the fact that the time of origin is not known, or in other words, that the observed tree does not include the stalk. The first option was to calculate the likelihood of the observed tree as the product of the likelihoods of its two subtrees, i.e.

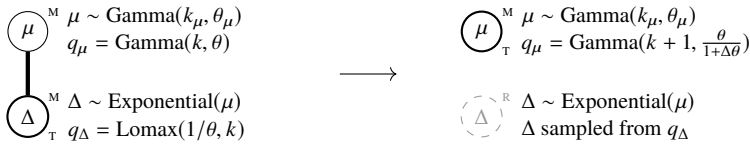
$$p(T|\theta) = p(T'_1|\theta)p(T'_2|\theta).$$



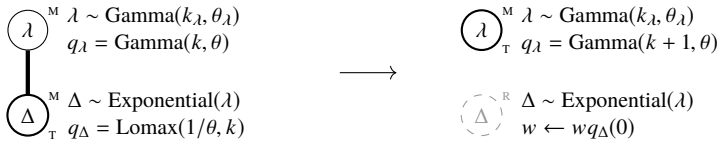
(a) Sampling c from $\text{Poisson}(\lambda\Delta)$.



(b) Observing 0 with respect to $\text{Poisson}(\mu\Delta)$.



(c) Sampling Δ from $\text{Exponential}(\mu)$.



(d) Observing 0 with respect to $\text{Exponential}(\lambda)$.

Figure 3.2: Delayed sampling in the phylogenetic birth-death models.

This option is exactly what we used so far in this chapter. We will now look closer at the second option: conditioning the posterior distribution on the time of the MRCA. Recall that the likelihood is in this case given by

$$p(T|\theta, t_{\text{MRCA}}) = \frac{p(T'_1|\theta)p(T'_2|\theta)}{S(\theta, t_{\text{MRCA}})^2}.$$

This implies that we need to divide the particle weight (the execution probability) by $S(\theta, t_{\text{MRCA}})^2$. For the CRBD model the survival probability is known analytically, but as mentioned before, this is not the case for most birth-death models.

In [I] we proposed a solution that does not require the survival probability to be known. After traversing the tree, we use the generative model to simulate two independent evolutionary processes starting at t_{MRCA} , and consider the outcome successful if both processes survive to the present time. We repeat this procedure until the first success, and multiply the weight of the particle by the number of trials (up to and including the successful one). This method is based on the fact that $1/S(\theta, t_{\text{MRCA}})^2$ is the expected value of the number of trials of a geometric distribution with the success probability $S(\theta, t_{\text{MRCA}})^2$:

$$\frac{1}{S(\theta, t_{\text{MRCA}})^2} = \sum_{M=1}^{\infty} M(1 - S(\theta, t_{\text{MRCA}})^2)^{M-1} S(\theta, t_{\text{MRCA}})^2.$$

The updated probabilistic program for the CRBD model is shown in Algorithm 3.4 (p. 66).

Algorithm 3.4: A probabilistic program for the CRBD model with the survivorship bias correction.

```

1:  $\lambda \sim$  prior distribution for  $\lambda$ 
2:  $\mu \sim$  prior distribution for  $\mu$ 
3: for all  $r \in$  nodes of  $T$  do
4:   if  $r$  is the root then
5:     continue
6:   end if
7:    $c_{\text{hs}} \sim \text{Poisson}(\lambda\Delta_r)$ 
8:   for  $i \leftarrow 1$  to  $c_{\text{hs}}$  do
9:      $t \sim \text{Uniform}(t_r, t_r + \Delta_r)$ 
10:    if not GOESEXTINCT( $t, \lambda, \mu$ ) then
11:      factor 0
12:    end if
13:    factor 2
14:  end for
15:  observe 0  $\sim \text{Poisson}(\mu\Delta_r)$ 
16:  if  $r$  is a speciation then
17:    observe 0  $\sim \text{Exponential}(\lambda)$ 
18:  end if
19: end for
20:  $M \leftarrow 1$ 
21: while GOESEXTINCT( $t_{\text{MRCA}}, \lambda, \mu$ ) or GOESEXTINCT( $t_{\text{MRCA}}, \lambda, \mu$ ) do
22:    $M \leftarrow M + 1$ 
23: end while
24: factor  $M$ 
25: return  $\lambda, \mu$ 
26: function GOESEXTINCT( $t, \lambda, \mu$ )
27:    $\Delta \sim \text{Exponential}(\mu)$ 
28:   if  $\Delta \geq t$  then
29:     return false
30:   end if
31:    $c_b \sim \text{Poisson}(\lambda\Delta)$ 
32:   for  $i \leftarrow 1$  to  $c_b$  do
33:      $t' \sim \text{Uniform}(t - \Delta, t)$ 
34:     if not GOESEXTINCT( $t', \lambda, \mu$ ) then
35:       return false
36:     end if
37:   end for
38:   return true
39: end function

```

Conclusion

This thesis is based on a collection of papers that introduce probabilistic programming as a completely new approach to parameter inference for phylogenetic birth-death models, and prove the feasibility of this approach. The birth-death models can be written as simple programs in probabilistic programming languages, and benefit from automatic inference which is an integral part of these languages. These programs are simple enough to allow biologists with just a basic understanding of programming to prototype, test and employ new models quickly, and without the need to derive and implement a bespoke inference algorithm.

With automatic inference based on sequential Monte Carlo methods, our approach, unlike the existing ones, also allows the model evidence to be estimated without bias. This in turn enables the employment of hierarchical inference algorithms as well as comparing different models.

The contribution to the automatic inference methods, namely delayed sampling and the extended alive particle filter, are not only relevant for phylogenetics, but for probabilistic programming in general. We have implemented well-known phylogenetic models and shown how the above-mentioned methods improve the quality of inference for these models.

Future work

Our work is not finished by any means. We have already mentioned some of the possible future directions in the previous chapter, namely the rejection control particle filter (and the question of how to determine its thresholds) and the question of finding the optimal traversal order of the observed tree.

In order to be able to work with big trees (with thousands of extant species) we

need to make the inference even more efficient (in terms of lowering the variance of the estimators and making the inference faster). This might include extending the existing inference algorithms as well as developing new algorithms and heuristics. Solving these problems is relevant not only for phylogenetics, but also for automatic inference in probabilistic programming languages in general.

In the present work we have assumed that the observed tree is known and mentioned briefly that it is reconstructed from such data as morphological traits and genomic data. We would like to join these two processes into one and employ probabilistic programming to infer both the tree and the parameters directly from these data.

Another interesting direction is solving the problem introduced in the beginning of the previous chapter: rather than implementing a generative program and just use a single observe, we needed to implement plenty of “partial” observes. How can we get PPLs to do this automatically?

We hope that our work will encourage other researchers and software developers to join us on our path towards the ultimate goal: to give computational phylogeneticists a set of tools that will enable them to think big about new models and reduce the time needed from an initial idea to running efficient simulation and inference for these models.

Acknowledgments

First, I would like to express my gratitude to my fantastic supervisors: Johannes Borgström, Thomas B. Schön and Lawrence M. Murray. A very special thanks to Lawrence: although not initially one of my supervisors, he always found time for me. Our long sessions, brainstorming and discussing ideas, were probably the most exciting time at Uppsala University for me.

A lot of the work included in this thesis has been done in cooperation with researchers from KTH Royal Institute of Technology and Swedish Museum of Natural History. I have always looked forward to our fortnightly meetings in Kista. Big thank you, Daniel Lundén, David Broman, Fredrik Ronquist, Viktor Senderov and Nicolas Lartillot! In addition to my supervisors, Fredrik and Victor also provided valuable feedback on the introductory chapters of this thesis.

Thanks to all my colleagues at the department, especially to the members of the concurrency group: Björn Victor, Lars-Henrik Eriksson, Arve Gengelbach, Tjark Weber, Anke Stüber and Joachim Parrow, as well as the former members Johannes Åman Pohjola, Ramunas Gutkovas and Sophia Knight; and to the members of the the Assemble group.

Thanks to Matteo Magnani for the experience to teach with him, and for the opportunity to take over the Database Design I course after he became the director of the undergraduate studies.

Thanks to all my unfailing lunch buddies: Daniel Kovacs and his colleagues from the chemistry department, Jan Ruzs, Lucia Komendova, Jorge Cayao, Arve and Anke. Thanks to all the friends I have got in Uppsala and who made these five years memorable!

Thanks to everybody who helped me, taught me something new and useful, or made me laugh, and whom I might have forgotten!

My work was supported by the Swedish Research Council via the grant no. 2013-4853 and by the Swedish Foundation for Strategic Research (SSF) via the project ASSEMBLE (contract number RIT15-0012).

Last but definitely not least, I would like to thank Viera, my wife, for all her support.

Summary in Swedish

Fylogenetiska födelse-/döds-modeller är en familj av matematiska modeller av evolution där man betraktar *speciering* (när populationen av en art delas upp i två grupper och så småningom bildar två nya arter) och *utrotning* (när hela populationen av en art dör ut) som plötsliga händelser som inträffar vid slumpmässiga tidpunkter. Med tanke på hur lång evolutionen är är ju en sådan förenkling acceptabel. Tiden det tar för en art att genomgå en speciering, respektive tills de utrotas, modelleras som exponentialfördelade slumpmässiga variabler. I de flesta modeller är hastighetsparametrarna inte konstanta: det kan handla om både kontinuerliga och plötsliga förändringar som påverkar en enskild art, grupper av arter eller alla arter. Vilka förändringar som är tillåtna och på vilket sätt dessa modelleras beror på de konkreta födelse-/döds-modellerna. Mer avancerade modeller kan också innehålla olika tillstånd för arter (t.ex. om en art är monogam eller inte) samt modellera deras förändringar.

Att använda fylogenetiska födelse-/döds-modeller för att simulera evolutionära processer på en dator är ganska enkelt: med enbart grundläggande programmeringskunskaper kan man implementera en födelse-/döds-modell som ett datorprogram. Vid körning börjar programmet med en enda art (vid en given tidpunkt i det förflutna) och simulerar steg för steg specierings- och utrotningshändelser. Ett resultat av en sådan simulering är en så kallad fullständig fylogeni (eller ett fullständigt evolutionärt träd). Med fullständig menas här att trädet också inkluderar utdöda arter. Det är lätt att konvertera detta träd till ett ”rekonstruerat” träd som bara visar utvecklingen för de befintliga arterna och deras förfäder – vi behöver bara ta bort de delar av den fullständiga fylogenin som endast är kopplade till de utdöda arterna.

Det omvända problemet är mer intressant och betydelsefullt för fylogenetiker: givet en fylogeni rekonstruerad från tillgängliga data om nu levande arter (t.ex. genomiska sekvenser) och en födelse-/döds-modell, vad kan man säga

om parametrarna för denna modell, t.ex. specierings- och utrotningshastigheter? För att svara på sådana frågor när forskare har föreslagit nya födelse-/dödsmodeller krävs det idag hjälp av externa experter. De hjälper till med att hitta och implementera skraddarsydda inferensalgoritmer, samt att implementera dem i befintliga eller nya programvarupaket som vanligtvis är ganska komplexa.

Vårt arbete löser detta problem på ett helt nytt sätt genom att använda så kallade *probabilistiska programmeringsspråk*. Dessa språk har inbyggt stöd för slumpmässiga variabler, sannolikhetsfördelningar och olika probabilistiska konstruktioner (som t.ex. sampling och observationer). En mycket viktig del är integrerad automatisk inferens: man behöver inte hitta och implementera någon inferensalgoritm (vilket vanligtvis är svårt) utan snarare beskriver man den generativa modellen i form av ett program som kan simulera denna modell (vilket vanligtvis är enkelt), lägger till informationen om vad som observerades, och kör programmet för att genomföra inferens automatiskt.

I denna doktorsavhandling visade vi hur man kan koda olika fylogenetiska födelse-/dödsmodeller som enkla program i probabilistiska programmeringsspråk och bevisat att det går att få användbara resultat inom rimlig tid. Vår metod är baserad på augmentation: programmet går igenom den rekonstruerade fylogenin, gren för gren, och lägger till obemärkta händelser och simulerar utvecklingen av utdöda arter. På detta sätt kan man uppskatta (fördelningar av) modellparametrarna.

Med automatisk inferens baserad på sekventiell Monte Carlo tillåter vårt tillvägagångssätt, till skillnad från de befintliga sätten, också att uppskatta marginell sannolikhet (sannolikhet att modellen kunde leda till just den aktuella rekonstruerade fylogenin) utan bias och möjliggör därmed användning av hierarkiska modeller samt jämförelse av olika modeller. Våra bidrag till de automatiska inferensmetoderna (t.ex. fördröjd sampling och det utvidgade *alive particle filter*) är inte relevanta endast för fylogenetik utan även för probabilistisk programmering generellt.

Bibliography

- [1] C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *Annals of Statistics*, 37(2):697–725, 2009.
- [2] C. Andrieu, A. Doucet, and V. B. Tadic. On-line parameter estimation in general state-space models. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 332–337, Seville, Spain, 2005.
- [3] C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 72(3): 269–342, 2010.
- [4] A. W. Appel and T. Jim. Continuation-passing, closure-passing style. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 293–302, Austin, Texas, USA, 1989.
- [5] 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.
- [6] 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.
- [7] R. E. Caflisch et al. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 1998:1–49, 1998.
- [8] 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.
- [9] O. F. Cook. Factors of species-formation. *Science*, 23(587):506–507, 1906.

- [10] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 221–236, Phoenix, Arizona, USA, 2019.
- [11] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859.
- [12] P. Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer–Verlag, New York, 2004.
- [13] 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.
- [14] A. Doucet, N. De Freitas, and N. Gordon. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo Methods in Practice*, pages 3–14. Springer, 2001.
- [15] A. J. Drummond and A. Rambaut. BEAST: Bayesian evolutionary analysis by sampling trees. *BMC Evolutionary Biology*, 7(1):214, 2007.
- [16] W. Feller. Die Grundlagen der Volterraschen Theorie des Kampfes ums Dasein in wahrscheinlichkeitstheoretischer Behandlung. *Acta Biotheoretica*, 5(1):11–40, May 1939.
- [17] R. G. FitzJohn. Quantitative traits and diversification. *Systematic Biology*, 59(6):619–633, 2010.
- [18] R. G. FitzJohn. Diversitree: comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution*, 3(6):1084–1092, 2012.
- [19] H. Ge, K. Xu, and Z. Ghahramani. Turing: a language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics*, pages 1682–1690, Playa Blanca, Lanzarote, Spain, 2018.
- [20] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984.
- [21] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

- [22] E. E. Goldberg, L. T. Lancaster, and R. H. Ree. Phylogenetic inference of reciprocal effects between geographic range evolution and diversification. *Systematic Biology*, 60(4):451–465, 2011.
- [23] N. D. Goodman and A. Stuhlmüller. The design and implementation of probabilistic programming languages. <http://dippl.org>, 2014. Accessed: 2021-2-1.
- [24] 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 on Uncertainty in Artificial Intelligence*, pages 220–229, Arlington, Virginia, USA, 2008.
- [25] J. M. Hammersley and D. C. Handscomb. Monte Carlo methods. *John Wiley & Sons, New York, USA*, 1964.
- [26] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
- [27] S. Höhna, W. A. Freyman, Z. Nolen, J. P. Huelsenbeck, M. R. May, and B. R. Moore. A Bayesian approach for estimating branch-specific speciation and extinction rates. Preprint at <https://www.biorxiv.org/content/10.1101/555805v1>, 2019.
- [28] J. P. Huelsenbeck and F. Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.
- [29] 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.
- [30] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.
- [31] D. G. Kendall. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics*, 19(1):1–15, 1948.
- [32] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
- [33] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. In *International Conference on Artificial Intelligence and Statistics*, pages 1338–1348, Fort Lauderdale, Florida, USA, 2017.

- [34] P. W. Lewis and G. S. Shedler. Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.
- [35] 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.
- [36] O. Maliet, F. Hartig, and H. Morlon. A model with many small shifts for estimating species-specific diversification rates. *Nature Ecology & Evolution*, 3(7):1086–1092, Jul 2019.
- [37] V. Mansinghka, R. Tibbetts, J. Baxter, P. Shafto, and B. Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. Preprint at <https://arxiv.org/abs/1512.05006>, 2015.
- [38] V. K. Mansinghka, D. Selsam, and Y. N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. Preprint at <https://arxiv.org/abs/1404.0099>, 2014.
- [39] 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.
- [40] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1352–1359, San Francisco, CA, USA, 2005.
- [41] D. Moen and H. Morlon. Why does diversification slow down? *Trends in Ecology & Evolution*, 29(4):190–197, 2014.
- [42] B. R. Moore, S. Höhna, M. R. May, B. Rannala, and J. P. Huelsenbeck. Critically evaluating the theory and performance of Bayesian analysis of macroevolutionary mixtures. *Proceedings of the National Academy of Sciences*, 113(34):9569–9574, 2016.
- [43] H. Morlon, E. Lewitus, F. L. Condamine, M. Manceau, J. Clavel, and J. Drury. RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution*, 7(5):589–597, 2016.
- [44] L. M. Murray. Bayesian state-space modelling on high-performance hardware using LibBi. *Journal of Statistical Software*, 67(10):1–36, 2015.

- [45] L. M. Murray. Lazy object copy as a platform for population-based probabilistic programming. Preprint at <https://arxiv.org/abs/2001.05293>, 2020.
- [46] L. M. Murray and T. B. Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43, 2018.
- [47] L. M. Murray, A. Lee, and P. E. Jacob. Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805, 2016.
- [48] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei. Variational sequential Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 968–977, Playa Blanca, Lanzarote, Spain, 2018.
- [49] P. Narayanan, J. Carette, W. Romano, C. Shan, and R. Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming*, pages 62–79, Kochi, Japan, 2016. Springer.
- [50] R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.
- [51] B. Paige and F. Wood. A compilation target for probabilistic programming languages. In *International Conference on Machine Learning*, pages 1935–1943, Beijing, China, 2014.
- [52] A. Pfeffer. *Practical Probabilistic Programming*. Manning, 2016.
- [53] M. Plummer. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, volume 124, pages 1–10. Vienna, Austria, 2003.
- [54] D. L. Rabosky. Automatic detection of key innovations, rate shifts, and diversity-dependence on phylogenetic trees. *PLoS ONE*, 9(2):e89543, 2014.
- [55] T. Stadler. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of Theoretical Biology*, 261(1):58–66, Nov. 2009.
- [56] T. Stadler. Mammalian phylogeny reveals recent diversification rate shifts. *Proceedings of the National Academy of Sciences*, 108(15):6187–6192, 2011.

- [57] 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.
- [58] A. Todeschini, F. Caron, M. Fuentes, P. Legrand, and P. Del Moral. Biips: Software for Bayesian inference with interacting particle systems. Preprint at <https://arxiv.org/abs/1412.3779>, 2014.
- [59] 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, Leuven, Belgium, 2016.
- [60] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. Preprint at <https://arxiv.org/abs/1610.09787>, 2016.
- [61] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. An introduction to probabilistic programming. Preprint at <https://arxiv.org/abs/1809.10756>, 2018.
- [62] J. Von Neumann. Various techniques used in connection with random digits. *Applied Mathematic Series*, 12(36-38):5, 1951.
- [63] D. Wingate and T. Weber. Automated variational inference in probabilistic programming. Preprint at <https://arxiv.org/abs/1301.1299>, 2013.
- [64] G. U. Yule. A mathematical theory of evolution, based on the conclusions of Dr. JC Willis, FRS. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character*, 213: 21–87, 1924.

Used distributions and their parameterizations

Bernoulli distribution

Notation: $k \sim \text{Bernoulli}(p)$

Parameters:

- probability $p \in [0, 1]$

Probability mass function:

$$f(k|p) = \begin{cases} p & \text{if } k = 1 \text{ (true)} \\ 1 - p & \text{if } k = 0 \text{ (false)} \end{cases}$$

Binomial distribution

Notation: $k \sim \text{Binomial}(n, p)$

Parameters:

- number of trials $n \in \mathbb{N} \cup \{0\}$
- probability of success $p \in [0, 1]$

Probability mass function:

$$f(k|n, p) = \binom{n}{k} p^k (1 - p)^{n-k} \text{ for } k \in \mathbb{N} \cup \{0\}$$

Exponential distribution

Notation: $x \sim \text{Exponential}(\lambda)$

Parameters:

- rate $\lambda > 0$

Probability density function:

$$f(x|\lambda) = \lambda e^{-\lambda x} \text{ for } x \geq 0$$

Gamma distribution

Notation: $x \sim \text{Gamma}(k, \theta)$

Parameters:

- shape $k > 0$
- scale $\theta > 0$

Probability density function:

$$f(x|k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta} \text{ for } x > 0$$

Lomax distribution

Notation: $x \sim \text{Lomax}(\lambda, \alpha)$

Parameters:

- scale $\lambda > 0$
- shape $\alpha > 0$

Probability density function:

$$f(x|\lambda, \alpha) = \frac{\alpha}{\lambda} \left(1 + \frac{x}{\lambda}\right)^{-(\alpha+1)} \text{ for } x \geq 0$$

Negative binomial distribution

Notation: $r \sim \text{NegativeBinomial}(k, p)$

Parameters:

- number of successes before the experiment is stopped $k \in \mathbb{N}$
- probability of success $p \in [0, 1]$

Probability mass function:

$$f(r|k, p) = \binom{r+k-1}{k-1} p^k (1-p)^r \text{ for } r \in \mathbb{N} \cup \{0\},$$

where r is the number of failures.

Normal (Gaussian) distribution

Notation: $x \sim \mathcal{N}(\mu, \sigma^2)$

Parameters:

- mean μ
- variance $\sigma^2 > 0$

Probability density function:

$$f(x|\mu, \sigma^2) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Poisson distribution

Notation: $k \sim \text{Poisson}(\lambda)$

Parameters:

- rate $\lambda > 0$

Probability mass function:

$$f(k|\lambda) = \frac{\lambda^k}{k!} e^{-\lambda} \text{ for } k \in \mathbb{N} \cup \{0\}$$

Uniform distribution

Notation: $x \sim \text{Uniform}(a, b)$

Parameters:

- lower bound a
- upper bound $b > a$

Probability density function:

$$f(x|a, b) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

B

Used abbreviations

APF Alive particle filter

BAMM Bayesian analysis of macro-evolutionary mixtures (model)

BiSSE Binary state speciation and extinction (model)

BP Before present (time)

BPF Bootstrap particle filter

ClaDS Cladogenetic diversification rate shift (model)

CPS Continuation-passing style

CPU Central processing unit

CRBD Constant-rate birth-death (model)

ESS Effective sample size

GPU Graphics processing unit

LSBDS Lineage-specific birth-death-shift (model)

MCMC Markov chain Monte Carlo

MRCA Most recent common ancestor

PF-RC Particle filter with rejection control

PGM Probabilistic graphical model

PMCMC Particle Markov chain Monte Carlo

PPL Probabilistic programming language

SMC Sequential Monte Carlo

Acta Universitatis Upsaliensis

*Digital Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology 2006*

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-432409



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2021

Paper I



Universal probabilistic programming offers a powerful approach to statistical phylogenetics

Fredrik Ronquist^{1†*}, Jan Kudlicka^{2†}, Viktor Senderov^{1†}, Johannes Borgström², Nicolas Lartillot³, Daniel Lundén⁴, Lawrence Murray⁵, Thomas B. Schön², David Broman⁴

¹Department of Bioinformatics and Genetics, Swedish Museum of Natural History, Box 50007, SE-104 05 Stockholm, Sweden

²Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden

³Laboratoire de Biométrie et Biologie Evolutive, UMR CNRS 5558, Université Claude Bernard Lyon 1, FR-69622 Villeurbanne Cedex, France

⁴Department of Computer Science, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden

⁵Uber AI, San Francisco CA 94105, United States

Statistical phylogenetic analysis currently relies on complex, dedicated software packages, making it difficult for evolutionary biologists to explore new models and inference strategies. Recent years have seen more generic solutions based on probabilistic graphical models, but this formalism can only partly express phylogenetic problems. Here we show that universal probabilistic programming languages (PPLs) solve the expressivity problem, while still supporting automated generation of efficient inference algorithms. To prove the latter point, we develop automated generation of sequential Monte Carlo (SMC) algorithms for PPL descriptions of arbitrary biological diversification (birth-death) models. SMC is a new inference strategy for these problems, supporting both parameter inference and efficient estimation of Bayes factors that are used in model testing. We take advantage of this in automatically generating SMC algorithms for several recent diversification models that have been difficult or impossible to tackle previously. Finally, applying these algorithms to 40 bird phylogenies, we show that models with slowing diversification, constant turnover and many small shifts generally explain the data best. Our work opens up several related problem domains to PPL approaches, and shows that few hurdles remain before these techniques can be effectively applied to the full range of phylogenetic models.

Introduction

In statistical phylogenetics, we are interested in learning the parameters of models in which evolutionary trees—phylogenies—play an important part. Such analyses have a surprisingly wide range of applications across the life sciences^{1,2,3}. In fact, the research front in many disciplines is partly defined today by our ability to learn the parameters of realistic phylogenetic models.

Statistical problems are often analyzed using generic modeling and inference tools. Not so in phylogenetics, where empiricists are largely dependent on dedicated software developed by small teams of computational biologists³. Even though these software packages have become increasingly flexible in recent years, empiricists are still limited to a large extent by predefined model spaces and inference strategies. Venturing outside these boundaries typically requires the help of skilled programmers and inference experts.

If it were possible to specify arbitrary phylogenetic models in an easy and intuitive way, and then automatically learn the latent variables (the unknown parameters) in them, the full creativity of the research community could be unleashed, significantly accelerating progress. There are two major hurdles standing in the way of such a vision. First, we must find a formalism (a language) that can express phylogenetic models in all their complexity, while still being easy to learn for empiricists (*the modeling language expressivity problem*). Second, we need to be able to generate computationally efficient inference algorithms from such model descriptions, drawing from the full range of techniques available today (*the automated inference problem*).

In recent years, there has been significant progress towards solving the expressivity problem by adopting the framework of probabilistic graphical models (PGMs)^{4,5}. PGMs

can express many components of phylogenetic models in a structured way, so that efficient Markov chain Monte Carlo (MCMC) samplers—the current workhorse of Bayesian statistical phylogenetics—can be automatically generated for them⁵. Other, more novel inference strategies are also readily applied to PGM descriptions of phylogenetic model components, as exemplified by recent work using STAN⁶ or the new Blang framework⁷.

Unfortunately, PGMs cannot express the core of phylogenetic models: the stochastic processes that generate the tree, and anything dependent on those processes. This is because the evolutionary tree has variable topology, while a PGM expresses a fixed topology. The problem even occurs on a fixed tree, if we need to express the possible existence of unobserved side branches that have gone extinct or have not been sampled. There could be any number of those for a given observed tree, each corresponding to a separate PGM instance.

Similar problems occur when describing evolutionary processes occurring on the branches of the tree. Many of the standard models considered today for trait evolution, such as continuous-time discrete-state Markov chains, are associated with an infinite number of possible change histories on a given branch. It is not always possible to represent this as a single distribution with an analytical likelihood that integrates out all change histories. Thus, it is sometimes necessary to describe the model as an unbounded stochastic loop or recursion over potential PGMs (individual change histories).

PGM-based systems may address these shortcomings by providing model components that hide underlying complexity. For instance, a tree may be represented as a single stochastic variable in a PGM-based model description⁵. An important disadvantage of this approach is that it removes information about complex model components from the model description. This forces users to choose among predefined alternatives instead of enabling them to describe how these model components are

*E-mail: fredrik.ronquist@nrm.se

†These authors contributed equally.

structured. Furthermore, computers can no longer ‘understand’ these components from the model description, making it impossible to automatically apply generic inference algorithms to them. Instead, special-purpose code has to be developed manually for each of the components. Finally, hiding a complex model component, such as a phylogenetic tree, also makes dependent variables unavailable for automated inference. In phylogenetics, for instance, a single stochastic tree node makes it impossible to describe branch-wise relations between the processes that generated the tree and other model components, such as the rate of evolution, the evolution of organism traits, or the dispersal of lineages across space.

Here, we show that the expressivity problem can be solved using universal probabilistic programming languages (PPLs). A *universal PPL* is an extension of a Turing-complete general-purpose language, which can express models with an unbounded number of random variables. This means that random variables are not fixed statically in the model (as they are in a finite PGM) but can be created dynamically during execution.

PPLs have a long history in computer science⁸, but until recently they have been largely of academic interest because of the difficulty of generating efficient inference machinery from model descriptions using such expressive languages. This is now changing rapidly thanks to improved methods of automated inference for PPLs^{9,10,11,12,13,14}, and the increased interest in more flexible approaches to statistical modeling and analysis. Current PPL inference algorithms provide state-of-the-art performance for many models but they are still quite inefficient for others. Improving PPL algorithms so that they can compete with manually engineered solutions for more problem domains is currently a very active research area.

To demonstrate the potential of PPLs in statistical phylogenetics, we tackle a tough problem domain: models that accommodate variation across lineages in diversification rate. These include the recent ClaDS (ClaDS0, ClaDS1, and ClaDS2)¹⁵, LSBDS¹⁶ and BAMB¹⁷ models, attracting considerable interest among evolutionary biologists despite the difficulties in developing good inference algorithms for some of them¹⁸.

Using WebPPL—an easy-to-learn PPL⁹—and Birch—a language with a more computationally efficient inference machinery¹⁴—we develop techniques that allow us to automatically generate efficient sequential Monte Carlo (SMC) algorithms from short descriptions of these models (~ 100 lines of code each). Although we found it convenient to work with WebPPL and Birch for this paper, we emphasize that similar work could have been done using other universal PPLs. Adopting the PPL approach allows us to generate the first efficient SMC algorithms for these models, and the first asymptotically exact inference machinery for the full BAMB model. Among other benefits, SMC inference allows us to directly estimate the marginal likelihoods of the models, so that we can assess their performance in explaining empirical data using rigorous Bayesian model comparison. Taking advantage of this, we show that models with slowing diversification, constant turnover and many small shifts (all combined in ClaDS2) generally explain the data from 40 bird phylogenies better than alternative models. We end the paper by discussing a few problems, all seemingly tractable, which remain to be solved before PPLs can be used to address the full range of phylogenetic models. Solving them would facilitate the adoption of a wide range of novel inference strategies that have seen little or no use in phylogenetics before.

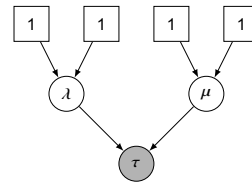


Fig. 1 A probabilistic graphical model describing constant rate birth-death (CRBD). The square boxes are fixed nodes (parameters of the gamma distributions) and the circles are random variables. The shaded variable (τ) is observed, and (λ, μ) are latent variables to be inferred.

Results

Probabilistic programming. Consider one of the simplest of all diversification models, constant rate birth-death (CRBD), in which lineages arise at a rate λ and die out at a rate μ , giving rise to a phylogenetic tree τ . Assume that we want to infer the values of λ and μ given some phylogenetic tree τ_{obs} of extant (now living) species that we have observed (or inferred from other data). In a Bayesian analysis, we would associate λ and μ with prior distributions, and then learn their joint posterior probability distribution given the observed value of τ .

Let us examine a PGM description of this model, say in RevBayes⁵ (Algorithm 1). The first statement in the description associates an observed tree with the variable `myTree`. The priors on `lambda` and `mu` are then specified, and it is stated that the tree variable `tau` is drawn from a birth-death process with parameters `lambda` and `mu` and generating a tree with leaves matching the taxa in `myTree`. Finally, `tau` is associated with (‘clamped to’) the observed value `myTree`.

Algorithm 1 PGM description of the CRBD model

```

1 myTree = readTrees("treefile.nex")
2
3 lambda ~ dnGamma(1, 1)
4 mu ~ dnGamma(1, 1)
5
6 tau ~ dnBirthDeath(lambda, mu, myTree.taxa)
7 tau.clamp(myTree)

```

There is a one-to-one correspondence between these statements and elements in the PGM graph describing the conditional dependencies between the random variables in the model (Figure 1). Given that the conditional densities `dnGamma` and `dnBirthDeath` are known analytically, along with good samplers, it is now straightforward to automatically generate standard inference algorithms, such as MCMC, for this problem.

Unfortunately, a PGM cannot describe from first principles (elementary probability distributions) how the birth-death process produces a tree of extant species. The PGM has a fixed graph structure, while the probability of a surviving tree is an integral over many outcomes with varying topology. Specifically, the computation of `dnBirthDeath` requires integration over all possible ways in which the process could have generated side branches that eventually go extinct, each of these with a unique configuration of speciation and extinction events (Figure 2). The integral must be computed by special-purpose code based on analytical or numerical solutions specific to the model. For the CRBD model, the integral is known analytically, but as

soon as we start experimenting with more sophisticated diversification scenarios, as evolutionary biologists would want to do, computing the integral is likely to require dedicated numerical solvers, if it can be computed at all.

Universal PPLs solve the expressivity problem by providing additional expressive power over PGMs. A PPL model description is essentially a simulation program (or generative model). Each time the program runs, it generates a different outcome. If it is executed an infinite number of times, we obtain a probability distribution over outcomes. Thus, a PPL provides a *programmatically model description*¹⁴.

A universal PPL provides two special constructs, one for drawing a random variable from a probability distribution, and one for conditioning a random variable on observed data. These special constructs are used by the PPL inference algorithms to manipulate executions of the program during inference. Many PPLs are embedded in existing programming languages, with these two special constructs added.

To use this approach, we need to write a PPL program so that the distribution over outcomes corresponds to the posterior probability distribution of interest. This is straightforward if we understand how to simulate from the model, and how to insert the constraints given by the observed data.

Assume, for instance, that we are interested in computing the probability of survival and extinction under CRBD for specific values of λ and μ , given that the process started at some time t in the past. We will pretend that we do not know the analytical solution to this problem; instead we will use a PPL to solve it. WebPPL⁹ is an easy-to-learn PPL based on JavaScript, and we will use it here for illustrating PPL concepts. WebPPL can be run in a web browser at <http://webppl.org> or installed locally (Supplementary Section 2.1). In WebPPL, the two special constructs mentioned above are: (1) the `sample` statement, which specifies the prior distributions from which random variables are drawn; and (2) the `condition` statement, conditioning a random variable on an observation. WebPPL provides a couple of alternatives to the `condition` statement, namely the `observe` and `factor` statements. These are explained in Supplementary Section 3.3.

In WebPPL, we define a function `goesExtinct`, which takes the values of time, λ and μ corresponding to variables t , λ and μ , respectively (Algorithm 2). It returns `true` if the process does not survive until the present (that is, goes extinct) and `false` otherwise (survives to the present).

Algorithm 2 Basic birth-death model simulation in WebPPL

```

1 var goesExtinct = function(time, lambda, mu) {
2   var waitingTime = sample(
3     Exponential({a: lambda + mu})
4   )
5
6   if (waitingTime > time) { return false }
7
8   var isSpeciation = sample(
9     Bernoulli({p: lambda / (lambda + mu)})
10  )
11
12  if (isSpeciation == false) { return true }
13
14  return goesExtinct(time - waitingTime, lambda, mu)
15    && goesExtinct(time - waitingTime, lambda, mu)
16 }

```

The function starts at some time $t > 0$ in the past. The `waitingTime` until the next event is drawn from an exponential distribution with rate $\lambda + \mu$ and compared with `time`. If `waitingTime > time`, the function returns `false` (the process survived). Otherwise, we flip a coin (the Bernoulli distribution) to determine whether the next event is a speciation or an extinction event. If it is a speciation, the process continues by calling the same function recursively for each of the daughter lineages with the updated time `time - waitingTime`. Otherwise the function returns `true` (the lineage went extinct).

If executed many times, the `goesExtinct` function defines a probability distribution on the outcome space `{true, false}` for specific values of t , λ and μ . To turn this into a Bayesian inference problem, let us associate λ and μ with gamma priors, and then infer the posterior distribution of these parameters assuming that we have observed a group originating at time $t = 10$ and surviving to the present. To do this, we combine the prior specifications and the conditioning on survival to the present with the `goesExtinct` function into a program that defines the distribution of interest (Algorithm 3).

Algorithm 3 CRBD model description in WebPPL

```

1 var model = function() {
2   var lambda = sample(
3     Gamma({shape: 1, scale: 1})
4   )
5   var mu = sample(
6     Gamma({shape: 1, scale: 1})
7   )
8   var t = 10
9
10  condition(goesExtinct(t, lambda, mu) == false)
11
12  return [lambda, mu]
13 }

```

The `goesExtinct` function described above (Algorithm 2) uses unbounded stochastic recursion: the tree that we simulate in the program can in principle grow to infinite size. This effectively proves that the probabilistic programming language defining this model, if it is to be used to simulate extinct side branches, must be a universal PPL. This, in turn, implies that a language that solves the expressivity problem in phylogenetics can also describe any phylogenetic model from which we can simulate using an algorithm. Adopting this approach thus allows a clean separation of model specification from inference. Of course, automated inference procedures now face the problem of executing complex universal PPL models on hardware with physical constraints, such as limited memory size. How-

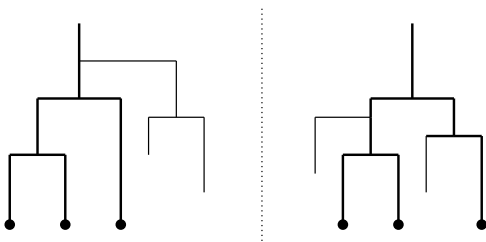


Fig. 2 Phylogenetic trees generated by a birth-death process. Two trees with extinct side branches (thin lines), each corresponding to the same observed phylogeny of extant species (thick lines). The trees illustrate just two examples of an infinite number of possible PGM expansions of the τ node in Figure 1.

A different perspective is represented by the cladogenetic diversification rate shift (ClADS) models¹⁵. They map diversification rate changes to speciation events, assuming that diversification rates change in small steps over the entire tree. After speciation, each descendant lineage inherits its initial speciation rate λ_i from the ending speciation rate λ_a of its ancestor through a mechanism that includes both a deterministic long-term trend and a stochastic effect. Specifically,

$$\log \lambda_i \sim \mathcal{N}(\log(\alpha \lambda_a), \sigma^2). \quad (2)$$

The α parameter determines the long-term trend, and its effects are similar to the z parameter of TDBD and BMM. When $\alpha < 1$, that is, $\log \alpha < 0$, the speciation rate of a lineage tends to decrease over time. The standard deviation σ determines the noise component. The larger the value, the more stochastic fluctuation there will be in speciation rates.

The original ClADS paper¹⁵ focuses on the rate multiplier $m = \alpha * \exp(\sigma^2/2)$ rather than on α , but we prefer the α parameterization mainly because it allows us to specify a conjugate prior that makes SMC inference more efficient (Supplementary Section 3). As pointed out elsewhere²⁵, the dynamics of the ClADS models is complex and differs considerably from superficially similar models, such as the BMM, TDBD and TDB models (for further discussion of this point, see Supplementary Section 3).

There are three different versions of ClADS, characterized by how they model μ . In ClADS0, there is no extinction, that is, $\mu = 0$. In ClADS1, there is a constant extinction rate μ throughout the tree. Finally, in ClADS2, it is the turnover rate $\epsilon = \mu/\lambda$ that is kept constant over the tree. All ClADS models collapse to CRB or CRBD models when $\alpha = 1$ and $\sigma \rightarrow 0$ (Figure 3). The ClADS models were initially implemented in the R package RPANDA²⁶, using a combination of advanced numerical solvers and MCMC simulation¹⁵. A new implementation of ClADS2 in Julia instead relies on data augmentation²⁵.

In contrast to previous work, where these models are implemented independently in complex software packages, we used PPL model descriptions (~ 100 lines of code each) to generate efficient and asymptotically correct inference machinery for all diversification models described above. The machinery we generate relies on SMC algorithms which, unlike classical MCMC, can also estimate the marginal likelihood (the normalization constant of Bayes theorem).

Estimating the marginal likelihood of a probability distribution that is only known up to a constant of proportionality is a hard problem in general. However, if we know how to sample from a similar distribution, classical importance sampling can provide a good estimate. The SMC algorithm is based on consecutive importance sampling from a series of probability distributions that change slowly towards the posterior distribution of interest. Thus, by piecing together the normalization constant estimates obtained in each of these steps, a good estimate of the marginal likelihood of the model is obtained essentially as a byproduct in the SMC algorithm^{27,28}. Such series of similar probability distributions are not available naturally in the MCMC algorithm, but have to be constructed in more involved, computationally complex procedures, such as thermodynamic integration^{29,30}, annealed importance sampling³¹ or stepping-stone sampling³².

Using the SMC machinery, we then compared the performance of the different diversification models on empirical

data by inferring the posterior distribution over the parameters of interest, and by conducting model comparison based on the marginal likelihood (Bayes factors). Specifically, we implemented the CRB, CRBD, TDB, TDBD, BMM, LSBDS, ClADS0, ClADS1 and ClADS2 models in WebPPL and Birch. The model descriptions are provided at <https://github.com/phypppl/probabilistic-programming>. They are similar in structure to the CRBD program presented above.

Inference strategies. We used inference algorithms in the SMC family, an option available in both WebPPL and Birch. An SMC algorithm^{33,34,35} runs many simulations (called particles) in parallel, and stops them when some new information, like the time of a speciation event or extinction of a side lineage, becomes available. At such points, the particles are subjected to *resampling*, that is, sampling (with replacement) based on their likelihoods. SMC algorithms work particularly well when the model can be written such that the information derived from observed data can successively be brought to bear on the likelihood of a particle during the simulation. This is the case when simulating a diversification process along a tree of extant taxa, because we know that each ‘hidden’ speciation event must eventually result in extinction of the unobserved side lineage. That is, we can condition the simulation on extinction of the side branches that arise (Supplementary Algorithm 3). Similarly, we can condition the simulation on the times of the speciation events leading to extant taxa.

Despite this, standard SMC (the bootstrap particle filter) remains relatively inefficient for these models, and is unlikely to yield adequate samples of the posterior for real problems given realistic computational budgets. Therefore, we employed three new PPL inference techniques that we developed or extended as part of this study: alignment³⁶, delayed sampling¹⁵ and the alive particle filter³⁷ (see Methods).

Empirical results. To demonstrate the power of the approach, we applied PPLs to compare the performance of the nine diversification models discussed above for 40 bird clades (see Methods and Supplementary Table 6). The results (Supplementary Figures 13–22) are well summarized by the four cases presented in Figure 4. Focusing on marginal likelihoods (top row), we observe that the simplest models (CRB, CRBD), without any variation through time or between lineages, provide an adequate description of the diversification process for around 40% of the trees (Figure 4 Alcedinidae). In the remaining clades, there is almost universal support for slowing diversification rates over time. Occasionally, this is not accompanied by strong evidence for lineage-specific effects (Figure 4 Muscicapidae+) but usually it is (Figure 4 Accipitridae and Lari). In the latter case, the ClADS models always show higher marginal likelihoods than BMM and LSBDS, and this even for trees on which the latter do detect rate shifts (Figure 4 Lari). Interestingly, ClADS2 rarely outperforms ClADS0, which assumes no extinction. More generally, models assuming no extinction often have a higher marginal likelihood than their counterparts allowing for it.

The parameter estimates (Figure 4) show the conservative nature of the Bayes factor tests, driven by the relatively vague priors we chose on the additional parameters of the more complex models (Supplementary Figure 2). However, even when complex models are marginally worse than simple or no-extinction models, there is evidence of the kind of variation they allow. For instance, the posterior distributions on z and $\log \alpha$ suggest

that negative time-dependence is quite generally present. Similarly, more sophisticated models usually detect low levels of extinction when they are outperformed by extinction-free counterparts. For a more extensive discussion of these and other results, see Supplementary Section 10.

Discussion

Universal PPLs provide stochastic recursion and dynamic creation of an unbounded number of random variables, which makes it possible to express virtually any interesting phylogenetic model. The expressiveness of PPLs is liberating for empiricists but it forces statisticians and computer scientists to approach the inference problem from a more abstract perspective. This can be challenging but also rewarding, as inference techniques for PPLs are so broadly applicable. Importantly, expressing phylogenetic models as PPLs opens up the possibility to apply a wide range of inference strategies developed for scientific problems with no direct relation to phylogenetics. Another benefit is that PPLs reduce the amount of manually written code for a particular inference problem, facilitating the task and minimizing the risk of inadvertently introducing errors, biases or inaccuracies. Our verification experiments (Supplementary Section 7) suggest that the light-weight PPL implementations of ClaDS1 and ClaDS2 provide more accurate computation of likelihoods than the thousands of lines of code developed in the initial implementation of these models¹⁵.

Previous discussion on the relative merits of diversification models have centered around the results of simulations and arguments over biological realism^{17,18,39,15,16}, and it has been complicated by the lack of asymptotically correct inference machinery for BAMM^{18,39}. Our most important contribution in this context is the refinement of PPL techniques so that it is now possible to implement correct and efficient parameter inference under a wide range of diversification models, and to compare their performance on real data using rigorous model testing procedures.

The PPL analyses of bird clades confirm previous claims that the ClaDS models provide a better description of lineage-specific diversification than BAMM¹⁵. Even when simpler models have higher likelihoods, the ClaDS models seem to pick up a consistent signal across clades of small, gradual changes in diversification rates. Like many previous studies⁴⁰, our analyses provide little or no support for extinction rates above zero. This might be due in part to systematic biases in the sampling of the leaves in the observed trees^{41,42}, a problem that could be addressed by extending our PPL model scripts (Supplementary Section 10.6). Such sampling biases can also give the impression of slowing diversification rates even when rates are constant, potentially explaining some of the support for negative values of z and $\log\alpha$ in our posterior estimates. We want to emphasize, however, that there is a range of other possible explanations for these patterns²³. The idea that lineage-specific variation in diversification rates might be responsible for low estimates of extinction rates in analyses using simpler models^{43,44} finds little support in our results but we cannot exclude the possibility that even more sophisticated lineage-specific models than the ones considered here might provide evidence in favor of this hypothesis. An interesting observation is that models with constant turnover (as in ClaDS2) appear to fit empirical data better than those with constant extinction (as in ClaDS1), even though

constant extinction has been commonly assumed in previous studies. A fascinating question that is now open to investigation is whether there remains evidence of occasional major shifts in diversification rates once the small gradual changes have been accounted for, something that could be addressed by a model that combines ClaDS- and BAMM-like features.

Our results show that PPLs can already now compete successfully with dedicated special-purpose software in several phylogenetic problem domains. Separately, we show how PPLs can be applied to models where diversification rates are dependent on observable traits of organisms (so-called state-dependent speciation and extinction models)³⁷. Other problem domains that may benefit from the PPL approach already at this point include epidemiology⁴⁵, host-parasite co-evolution⁴⁶, and biogeography^{47,48,49,50}.

What is missing before it becomes possible to generate efficient inference machinery for the full range of phylogenetic models from PPL descriptions? Assume, for instance, that we would like to do joint inference of phylogeny (say from DNA sequence data) and diversification processes, instead of assuming that the extant tree is observed. This would seem to touch on the major obstacles that remain. We then need to extend our current PPL models so that they also describe the nucleotide substitution process along the tree, and condition the simulation on the observed sequences. To generate the standard MCMC machinery for sampling across trees from such descriptions, delayed sampling needs to be extended to summarize over ancestral sequences (Felsenstein's pruning algorithm)⁵¹, and it should be applied statically through analysis of the script before the MCMC starts rather than dynamically. State-of-the-art MCMC algorithms for PPLs¹² must then be extended to generate computationally efficient tree samplers, such as stochastic nearest neighbour interchange⁵². Applying SMC algorithms for sampling across trees⁵³ is even simpler, it just requires delayed sampling to summarize over ancestral sequences. To facilitate use of PPLs, we think it will also be important to provide a domain-specific PPL that is easy to use, while supporting both automatic state-of-the-art inference algorithms for phylogenetic problems as well as manual composition of novel inference strategies suited for this application domain. These all seem to be tractable problems, which we aim to address within the TreePPL project (treepppl.org).

As the field of probabilistic programming is currently in a phase of intense experimentation, new PPL platforms—both universal and non-universal—are continuously presented and many existing ones are actively developed. Several of these platforms are likely to be useful for phylogenetic problems, not the least since they explore novel inference algorithms—such as automatic variational inference⁵⁴, adaptive Hamiltonian Monte Carlo⁵⁵, non-reversible parallel tempering⁵⁶ or sequential change of measure⁵⁷—that have only recently started to find their way into statistical phylogenetics^{58,6,59}. Interesting platforms include not only RevBayes⁵, specifically designed for phylogenetics, but also more general platforms such as STAN⁶⁰, Anglican¹⁰, PyMC3⁶¹, Edward⁶², Pyro⁶³ and Blang⁷. We think that evolutionary biologists exploring these new tools will be excited by the expressivity of universal PPLs and the generality across model space of the automated inference solutions designed for them. With this in mind, we invite readers with an interest in computational methods to join us and others in developing languages and inference strategies supporting this powerful new approach to statistical phylogenetics.

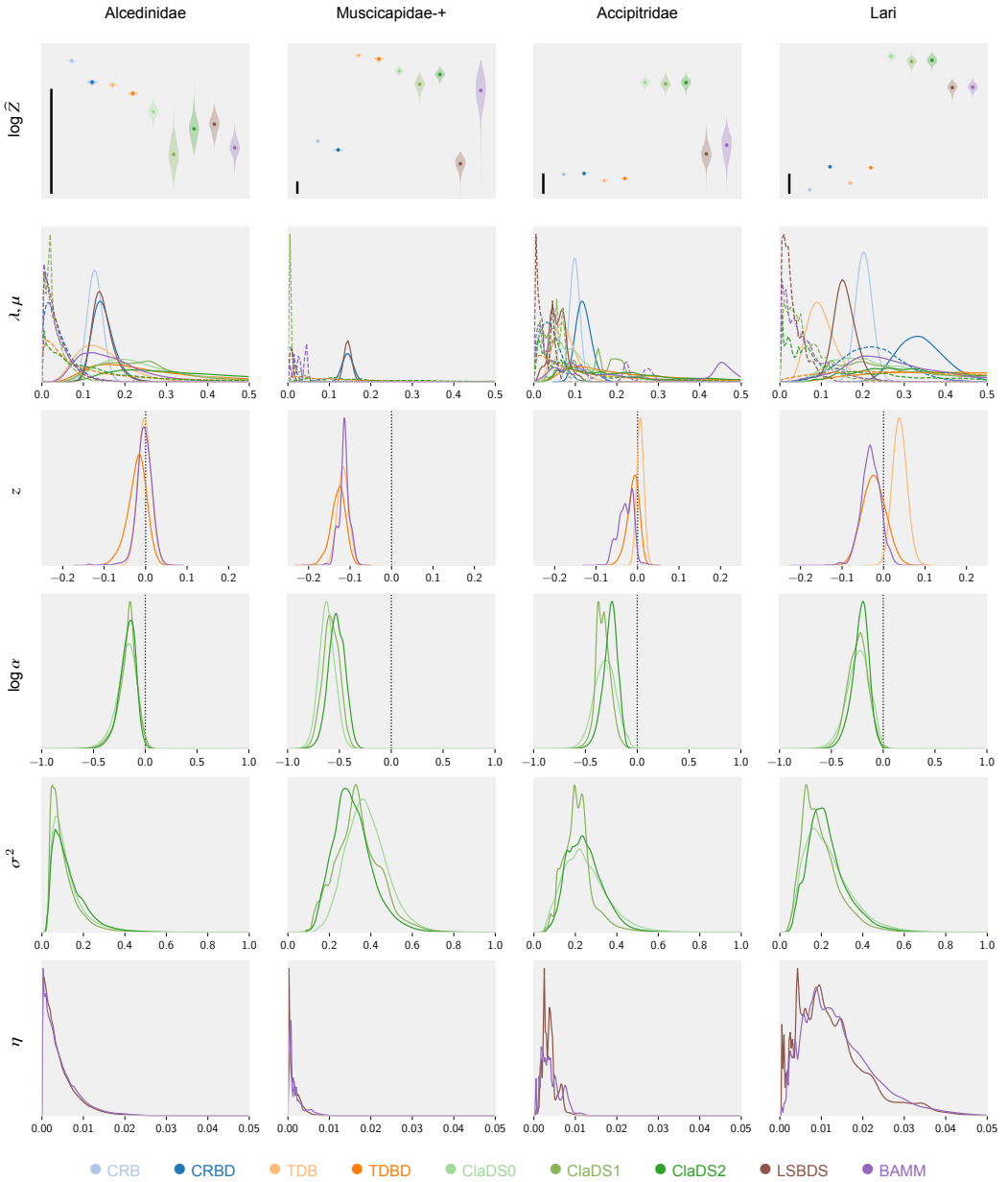


Fig. 4 Comparison of diversification models for four bird clades exemplifying different patterns. **Alcedinidae:** simple models are adequate; **Muscipidae+:** slowing diversification but no or weak lineage-specific effects; **Accipitridae:** gradual (ClaDS) lineage-specific changes in diversification; and **Lari:** evidence for both gradual (ClaDS) and for punctuated (BAMM and LSBDS) lineage-specific changes in diversification. The top row shows the estimated marginal likelihoods (log scale; violin plots with a dot marking the median estimate). A difference of 5 units (scale bar) is considered strong evidence in favor of the better model³⁸. The remaining rows show estimated posterior distributions for different model parameters specified along the left margin. The μ distributions are shown with dashed lines, all other distributions with unbroken lines. The colors represent different models (see legend).

Methods

PPL software and model scripts. All PPL analyses described here used WebPPL version 0.9.15, Node version 12.13.1⁹ and the most recent development version of Birch (as of June 12, 2020)¹⁴. We implemented all models (CRB, CRBD, TDB, TDBD, ClaDS0, ClaDS1, ClaDS2, LSBDS and BMM) as explicit simulation scripts that follow the structure of the CRBD example discussed in the main text (Supplementary Section 5). We also implemented compact simulations for the four simplest models (CRB, CRBD, TDB and TDBD) using the analytical equations for specific values of λ , μ and z to compute the probability of the observed trees.

In the PPL model descriptions, we account for incomplete sampling of the tips in the phylogeny based on the ρ -sampling model⁶⁴. That is, each tip is assumed to be sampled with a probability ρ , which is specified a priori. To simplify the presentation in this paper, we usually assume $\rho = 1$. However, the model scripts we developed support $\rho < 1$, and in the empirical analyses we set ρ for each bird tree to the proportion of the known species included in the tree.

We standardized prior distributions across models to facilitate model comparisons (Supplementary Section 4, Supplementary Figure 2). To simplify the scripts, we simulated outcomes on ordered but unlabeled trees, and reweighted the particles so that the generated density was correct for labelled and unordered trees (Supplementary Section 3.2). We also developed an efficient simulation procedure to correct for survivorship bias, that is, the fact that we can only observe trees that survive until the present (Supplementary Section 5.3).

Inference strategies. To make SMC algorithms more efficient on diversification model scripts, we applied three new PPL inference techniques: alignment, delayed sampling, and the alive particle filter. *Alignment*^{36,65} refers to the synchronization of resampling points across simulations (particles) in the SMC algorithm. The SMC algorithms previously used for PPLs automatically resample particles when they reach **observe** or **condition** statements. Diversification simulation scripts will have different numbers and placements of hidden speciation events on the surviving tree (Figure 2), each associated with a **condition** statement in a naive script. Therefore, when particles are compared at resampling points, some may have processed a much larger part of the observed tree than others. Intuitively, one would expect the algorithm to perform better if the resampling points were aligned, such that the particles have processed the same portion of the tree when they are compared. This is indeed the case; alignment is particularly important for efficient inference on large trees (Supplementary Figure 3). Alignment at code branching points (corresponding to observed speciation events in the diversification model scripts) can be generated automatically through static analysis of model scripts³⁶. Here, we manually aligned the scripts by replacing the statements that normally trigger resampling with code that accumulate probabilities when they did not occur at the desired locations in the simulation (Supplementary Section 6.1).

*Delayed sampling*¹³ is a technique that uses conjugacy to avoid sampling parameter values. For instance, the gamma distribution we used for λ and μ is a conjugate prior to the Poisson distribution, describing the number of births or deaths expected to occur in a given time period. This means that we can marginalize out the rate, and simulate the number of

events directly from its marginal (gamma-Poisson) distribution, without having to first draw a specific value of λ or μ . In this way, a single particle can cover a portion of parameter space, rather than just single values of λ and μ . Delayed sampling is only available in Birch; we extended it to cover all conjugacy relations relevant for the diversification models examined here.

The *alive particle filter*³⁷ is a technique for improving SMC algorithms when some particles can ‘die’ because their likelihood becomes zero. This happens when SMC is applied to diversification models because simulations that generate hidden side branches surviving to the present need to be discarded. The alive particle filter is a generic improvement on SMC, and it collapses to standard SMC with negligible overhead when no particles die. This improved version of SMC, partly inspired by our work on state-dependent speciation-extinction models³⁷, is only available in Birch.

Verification. To verify that the model scripts and the automatically generated inference algorithms are correct, we performed a series of tests focusing on the normalization constant (Supplementary Section 7). First, we checked that the model scripts for simple models (CRB(D) and TDB(D)) generated normalization constant estimates that were consistent with analytically computed likelihoods for specific model parameter values (Supplementary Figure 4). Second, we used the fact that all advanced diversification models (ClaDS0-2, LSBDS, BMM) collapse to the CRBD model under specific conditions, and verified that we obtained the correct likelihoods for a range of parameter values (Supplementary Figure 6). Third, we verified for the advanced models that the independently implemented model scripts and the inference algorithms generated for them by WebPPL and Birch, respectively, estimated the same normalization constant for a range of model parameter values (Supplementary Figure 7). Fourth, we checked that our normalization constant estimates were consistent with the RPANDA package^{26,15} for ClaDS0, ClaDS1, and ClaDS2, and with RevBayes for LSBDS^{5,16}. For these tests, we had to develop specialized PPL scripts emulating the likelihood computations of RPANDA and RevBayes. The normalization constant estimates matched for LSBDS (Supplementary Figure 9) and for ClaDS0 (Supplementary Figure 8); for ClaDS1 and ClaDS2, they matched for low values of λ and μ (or ϵ) but not for larger values (Supplementary Figure 8). Our best-effort interpretation at this point is that the PPL estimates for ClaDS1 and ClaDS2 are more accurate than those obtained from RPANDA for these values (Supplementary Section 7.4). Finally, as there is no independent software that computes BMM likelihoods correctly yet, we checked that our BMM scripts gave the same normalization constant estimates as LSBDS under settings where the former model collapses to the latter (Supplementary Figure 10).

Data. We applied our PPL scripts to 40 bird clades derived from a previous analysis of divergence times and relationships among all bird species⁶⁶. The selected clades are those with more than 50 species (range 54–316) after outgroups had been excluded (Supplementary Table 6). We followed the previous ClaDS2 analysis of these clades¹⁵ in converting the time scale of the source trees to absolute time units. The clade ages range from 12.5 Ma to 66.6 Ma.

Bayesian inference. Based on JavaScript, WebPPL is comparatively slow, making it less useful for high-precision com-

putation of normalization constants or estimation of posterior probability distributions using many particles. WebPPL is also less efficient than Birch because it does not yet support delayed sampling and the alive particle filter. Delayed sampling, in particular, substantially improves the quality of the posterior estimates obtained with a given number of particles. Therefore, we focused on Birch in computing normalization constants and posterior estimates for the bird clades.

For each tree, we ran the programs implementing the ClaDS, BAMB and LSBDS models using SMC with delayed sampling and the alive particle filter as the inference method. We ran each program 500 times and collected the estimates of $\log Z$ from each run together with the information needed to estimate the posterior distributions. The quality of the normalization constant estimates (on the log scale) from these 500 runs was estimated using the standard deviation, as well as the relative effective sample size and the conditional acceptance rate (Supplementary Section 9). We initially set the number of SMC particles to 5,000, which was sufficient to obtain high-quality estimates for all models except BAMB (Supplementary Table 7). We increased the number of particles to 20,000 for BAMB to obtain estimates of acceptable quality for this model.

For CRB, CRBD, TDB and TDBD we exploited the closed form for the likelihood in the programs. We used importance sampling with 10,000 particles as the inference method, and ran each program 500 times. This was sufficient to obtain estimates of very high accuracy for all models (Supplementary Table 7). The computational resources we used to obtain the results are specified in Supplementary Table 8.

Visualization. Visualizations were prepared with Matplotlib⁶⁷. We used the collected data from all runs to draw violin plots for $\log \hat{Z}$ as well as the posterior distributions for λ , μ (for all models), z (for TDB, TDBD and BAMB), $\log \alpha$ and σ^2 (for the ClaDS models), and η (for LSBDS and BAMB). By virtue of delayed sampling, the posterior distributions for λ and μ for all ClaDS models, as well as for BAMB and LSBDS, were calculated as mixtures of gamma distributions, the posterior distribution for $\log \alpha$ and σ^2 for all ClaDS models as mixtures of normal inverse gamma and inverse gamma distributions, and the posterior distribution for η for BAMB and LSBDS as a mixture of gamma distributions. For the remaining model parameters, we used the kernel density estimation (KDE) method. Exact plot settings and plot data are provided in the code repository accompanying the paper.

Reporting Summary Further information on research design is available in the Nature Research Reporting Summary linked to this article.

Data availability

The dated phylogenetic trees used to compare the diversification models, together with full literature references, can be found at <https://github.com/phypp1/probabilistic-programming>, under the directory `data`. Supplementary information is available at <https://github.com/phypp1/probabilistic-programming> under the directory `supplementary`.

Code availability

The WebPPL and Birch models can be found in the same repository, <https://github.com/phypp1/probabilistic-programming>, under the directories `webppl` and `birch`.

References

1. Felsenstein, J. *Inferring Phylogenies* (Sinauer Associates, Sunderland, Massachusetts, 2003).
2. Yang, Z. *Molecular Evolution: A Statistical Approach* (Oxford University Press, Oxford, United Kingdom; New York, NY, United States of America, 2014).
3. Nascimento, F. F., dos Reis, M. & Yang, Z. A biologists guide to Bayesian phylogenetic analysis. *Nature Ecology & Evolution* **1**, 1446–1454 (2017).
4. Höhna, S. et al. Probabilistic graphical model representation in phylogenetics. *Systematic Biology* **63**, 753–771 (2014).
5. Höhna, S. et al. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology* **65**, 726–736 (2016).
6. Fourment, M. & Darling, A. E. Evaluating probabilistic programming and fast variational Bayesian inference in phylogenetics. *PeerJ* **7**, e8272 (2019).
7. Bouchard-Côté, A. et al. Blang: Bayesian declarative modelling of arbitrary data structures. Preprint at <https://arxiv.org/abs/1912.10396> (2019).
8. Kozen, D. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science*, pages 101–114 (San Juan, Puerto Rico, USA, 1979).
9. Goodman, N. D. & Stuhlmüller, A. The design and implementation of probabilistic programming languages. <http://dippl.org> (2014). Accessed: 2020-5-12.
10. Wood, F., Meent, J. W. & Mansinghka, V. A new approach to probabilistic programming inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 1024–1032 (Reykjavik, Iceland, 2014).
11. Mansinghka, V., Selsam, D. & Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. Preprint at <https://arxiv.org/abs/1404.0099> (2014).
12. Ritchie, D., Stuhlmüller, A. & Goodman, N. C3: Lightweight incrementalized MCMC for probabilistic programs using continuations and callsite caching. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 28–37 (Cadiz, Spain, 2016).
13. Murray, L. M., Lundén, D., Kudlicka, J., Broman, D. & Schön, T. B. Delayed sampling and automatic RaoBlackwellization of probabilistic programs. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 21, page 10 (Lanzarote, 2018).

14. Murray, L. M. & Schön, T. B. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control* **46**, 29–43 (2018).
15. Maliet, O., Hartig, F. & Morlon, H. A model with many small shifts for estimating species-specific diversification rates. *Nature Ecology & Evolution* **3**, 1086–1092 (2019).
16. Höhna, S. et al. A Bayesian approach for estimating branch-specific speciation and extinction rates. Preprint at <https://biorxiv.org/content/10.1101/555805v1> (2019).
17. Rabosky, D. L. Automatic detection of key innovations, rate shifts, and diversity-dependence on phylogenetic trees. *PLoS ONE* **9**, e89543 (2014).
18. Moore, B. R., Höhna, S., May, M. R., Rannala, B. & Huelsenbeck, J. P. Critically evaluating the theory and performance of Bayesian analysis of macroevolutionary mixtures. *Proceedings of the National Academy of Sciences of the United States of America* **113**, 9569–9574 (2016).
19. Yule, G. U. A mathematical theory of evolution, based on the conclusions of Dr. J.C. Willis, FRS. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character* **213**, 21–87 (1924).
20. Nee, S. Birth-death models in macroevolution. *Annual Review of Ecology, Evolution and Systematics* **37**, 1–17 (2006).
21. Feller, W. Die Grundlagen der Volterraschen Theorie des Kampfes ums Dasein in wahrscheinlichkeitstheoretischer Behandlung. *Acta Biotheoretica* **5**, 11–40 (1939).
22. Kendall, D. G. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics* **19**, 1–15 (1948).
23. Moen, D. & Morlon, H. Why does diversification slow down? *Trends in Ecology & Evolution* **29**, 190–197 (2014).
24. Rabosky, D. L. et al. BAMMtools: an R package for the analysis of evolutionary dynamics on phylogenetic trees. *Methods in Ecology and Evolution* **5**, 701–707 (2014).
25. Maliet, O. & Morlon, H. Fast and accurate estimation of species-specific diversification rates using data augmentation. Preprint at <https://www.biorxiv.org/content/10.1101/2020.11.03.365155v1> (2020).
26. Morlon, H. et al. RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution* **7**, 589–597 (2016).
27. Hamze, F. & de Freitas, N. Hot Coupling: A particle approach to inference and normalization on pairwise undirected graphs. In Y. Weiss, B. Schölkopf & J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 491–498 (MIT Press, 2006).
28. Andersson Naesseth, C., Lindsten, F. & Schön, T. B. Sequential Monte Carlo for graphical models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence & K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1862–1870 (Curran Associates, Inc., 2014).
29. Gelman, A. & Meng, X.-L. Simulating normalizing constants: from importance sampling to bridge sampling to path sampling. *Statistical Science* **13**, 163–185 (1998).
30. Lartillot, N. & Philippe, H. Computing Bayes factors using thermodynamic integration. *Systematic Biology* **55**, 195–207 (2006).
31. Neal, R. M. Annealed importance sampling. *Statistics and Computing* **11**, 125–139 (2001).
32. Xie, W., Lewis, P. O., Fan, Y., Kuo, L. & Chen, M.-H. Improving marginal likelihood estimation for Bayesian phylogenetic model selection. *Systematic Biology* **60**, 150–160 (2011).
33. Doucet, A. & Johansen, A. A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan & B. Rozowski, editors, *The Oxford Handbook of Nonlinear Filtering*, chapter 24, pages 656 – 704 (Oxford University Press, Oxford, UK, 2008).
34. Doucet, A. & Lee, A. Sequential Monte Carlo methods. In M. Maathuis, M. Drton, S. Lauritzen & M. Wainwright, editors, *Handbook of Graphical Models*, chapter 7, pages 165 – 188 (CRC Press, Boca Raton, FL, USA, 2019).
35. Naesseth, C. A., Lindsten, F. & Schön, T. B. Elements of Sequential Monte Carlo. *Foundations and Trends in Machine Learning* **12**, 307–392 (2019).
36. Lundén, D., Broman, D., Ronquist, F. & Murray, L. M. Automatic alignment of Sequential Monte Carlo inference in higher-order probabilistic programs. Preprint at <https://arxiv.org/abs/1812.07439> (2018).
37. Kudlicka, J., Murray, L. M., Ronquist, F. & Schön, T. B. Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence 2019*, volume 2019, page 11 (Tel Aviv, Israel, 2019).
38. Jeffreys, H. *The Theory of Probability* (Oxford University Press, Oxford, 1961).
39. Rabosky, D. L., Mitchell, J. S. & Chang, J. Is BAMM flawed? Theoretical and practical concerns in the analysis of multi-rate diversification models. *Systematic biology* **66**, 477–498 (2017).
40. Pyron, R. A. & Burbrink, F. T. Phylogenetic estimates of speciation and extinction rates for testing ecological and evolutionary hypotheses. *Trends in Ecology & Evolution* **28**, 729–736 (2013).
41. Höhna, S., Stadler, T., Ronquist, F. & Britton, T. Inferring speciation and extinction rates under different sampling schemes. *Molecular Biology and Evolution* **28**, 2577–2589 (2011).
42. Rosindell, J., Cornell, S. J., Hubbell, S. P. & Etienne, R. S. Protracted speciation revitalizes the neutral theory of biodiversity. *Ecology Letters* **13**, 716–727 (2010).
43. Rabosky, D. L. Extinction rates should not be estimated from molecular phylogenies. *Evolution* **64**, 1816–1824 (2010).

44. Morlon, H., Parsons, T. L. & Plotkin, J. B. Reconciling molecular phylogenies with the fossil record. *Proceedings of the National Academy of Sciences of the United States of America* **108**, 16327–16332 (2011).
45. Baele, G., Dellicour, S., Suchard, M. A., Lemey, P. & Vrancken, B. Recent advances in computational phylodynamics. *Current Opinion in Virology* **31**, 24–32 (2018).
46. Braga, M. P., Landis, M. J., Nylín, S., Janz, N. & Ronquist, F. Bayesian inference of ancestral host-parasite interactions under a phylogenetic model of host repertoire evolution. *Systematic Biology* **67**, 000–000 (Advance access) (2020).
47. Ronquist, F. & Sanmartín, I. Phylogenetic methods in biogeography. *Annual Review of Ecology, Evolution, and Systematics* **42**, 441–464 (2011).
48. Matzke, N. J. Model selection in historical biogeography reveals that founder-event speciation is a crucial process in island clades. *Systematic Biology* **63**, 951–970 (2014).
49. Landis, M. J., Matzke, N. J., Moore, B. R. & Huelsenbeck, J. P. Bayesian analysis of biogeography when the number of areas is large. *Systematic Biology* **62**, 789–804 (2013).
50. Ree, R. H. & Sanmartín, I. Conceptual and statistical problems with the DEC+J model of founder-event speciation and its comparison with DEC via model selection. *Journal of Biogeography* **45**, 741–749 (2018).
51. Felsenstein, J. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of molecular evolution* **17**, 368–376 (1981).
52. Lakner, C., van der Mark, P., Huelsenbeck, J. P., Larget, B. & Ronquist, F. Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics. *Systematic Biology* **57**, 86–103 (2008).
53. Bouchard-Côté, A., Sankararaman, S. & Jordan, M. I. Phylogenetic inference via Sequential Monte Carlo. *Systematic biology* **61**, 579–593 (2012).
54. Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A. & Blei, D. M. Automatic differentiation variational inference. Preprint at <http://arxiv.org/abs/1603.00788> (2016).
55. Hoffman, M. D. & Gelman, A. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *The Journal of Machine Learning Research* **15**, 1593–1623 (2014).
56. Syed, S., Bouchard-Côté, A., Deligiannidis, G. & Doucet, A. Non-reversible parallel tempering: a scalable highly parallel MCMC scheme. Preprint at <http://arxiv.org/abs/1905.02939> (2019).
57. Zhou, Y., Johansen, A. M. & Aston, J. A. Toward automatic model comparison: An adaptive Sequential Monte Carlo approach. *Journal of Computational and Graphical Statistics* **25**, 701–726 (2016).
58. Dinh, V., Bilge, A., Zhang, C. & Matsen, F. A. Probabilistic path Hamiltonian Monte Carlo. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1–10 (Sydney, Australia, 2017).
59. Wang, L., Wang, S. & Bouchard-Côté, A. An annealed Sequential Monte Carlo method for Bayesian phylogenetics. *Systematic Biology* **69**, 155–183 (2020).
60. Carpenter, B. et al. Stan: A probabilistic programming language. *Journal of Statistical Software* **76** (2017).
61. Salvatier, J., Wiecki, T. V. & Fonnesbeck, C. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science* **2**, e55 (2016).
62. Tran, D. et al. Edward: A library for probabilistic modeling, inference, and criticism. Preprint at <https://arxiv.org/abs/1610.09787> (2016).
63. Bingham, E. et al. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research* **20**, 1–6 (2019).
64. Stadler, T. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of Theoretical Biology* **261**, 58–66 (2009).
65. Lundén, D., Borgström, J. & Broman, D. Correctness of Sequential Monte Carlo inference for probabilistic programming languages. Preprint at <https://arxiv.org/abs/2003.05191> (2020).
66. Jetz, W., Thomas, G. H., Joy, J. B., Hartmann, K. & Mooers, A. O. The global diversity of birds in space and time. *Nature* **491**, 444–448 (2012).
67. Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **9**, 90–95 (2007).

Acknowledgements

We thank Lars Arvestad and Philippe Veber for their contributions to the early development of the ideas presented here, and Odile Maliet for extensive help with the RPANDA implementation of the ClaDS models. H el ene Morlon, Bret Larget and an anonymous reviewer provided many valuable comments that helped improve the manuscript. This work was supported by grants from the Swedish Research Council to J.B. and J.K. (No 2013-4853) and to F.R. (No 2018-04620), by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012) to T.S., D.B. and J.K., and by the European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement PhyPPL No 898120 to V.S. The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at the National Supercomputer Centre (NSC).

Contributions

F.R. and N.L. initiated the project. All authors contributed to the further development of concepts and algorithms. F.R., J.K. and V.S. implemented algorithms, supported by D.L., J.B., L.M., N.L., T.S. and D.B. Verification experiments and empirical analyses were run by J.K. and V.S., who also generated most of the illustrations assisted by D.L., F.R. and J.B. The final manuscript was a joint effort.

Correspondence and requests for materials should be addressed to F.R., J.K. or V.S.

Competing interests

The authors declare no competing interests.

Supplementary information for “Universal probabilistic programming offers a powerful approach to statistical phylogenetics”

Fredrik Ronquist^{1†*}, Jan Kudlicka^{2†}, Viktor Senderov^{1†}, Johannes Borgström², Nicolas Lartillot³, Daniel Lundén⁴, Lawrence Murray⁵, Thomas B. Schön², David Broman⁴

¹Department of Bioinformatics and Genetics, Swedish Museum of Natural History, Box 50007, SE-104 05 Stockholm, Sweden

²Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden

³Laboratoire de Biométrie et Biologie Evolutive, UMR CNRS 5558, Université Claude Bernard Lyon 1, FR-69622 Villeurbanne Cedex, France

⁴Department of Computer Science, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden

⁵Uber AI, San Francisco CA 94105, United States

1 Probabilistic programming: an introduction

In this section, we give a brief introduction to (universal) probabilistic programming languages (PPLs), focusing on the key constructs that are available in most PPLs. First, we give a short overview of different PPLs, followed by an introduction to three essential concepts in probabilistic programming: (i) sampling, (ii) conditioning, and (iii) inference. These concepts are illustrated here in the WebPPL language^a, which is based on a functional subset^b of the JavaScript language.

1.1 Overview

A central objective of probabilistic programming is to separate the *model* from the *inference* algorithm, such that a user can construct and use a probabilistic model without the need to implement the inference algorithm explicitly. Instead, it is the task of the runtime system of the PPL to automatically perform the inference, potentially based on some method preferences specified by the user.

Programming and modeling languages that separate the model specification from the inference algorithm have been around for several decades. One of the first of these languages is BUGS (Bayesian inference Using Gibbs Sampling)¹. BUGS allows users to describe probabilistic graphical models²—in particular Bayesian networks—in a declarative way. The model parameters of interest are then estimated by automatically applying Bayesian inference using Gibbs sampling (and some other methods). More recent languages that separate modeling and inference of graphical models include Infer.NET³.

Although the above-mentioned languages and environments have shown great success in their application areas, they have certain model restrictions. In particular, they are limited to models where the dependencies between random variables can be expressed as a Bayesian network, that is, a finite directed acyclic graph, potentially with if-then-else conditions over variables. In some domains, this is not sufficient to describe the models of interest. Rather recently, the concept of *probabilistic programming languages*⁴ has gained significant attention as a promising solution, in particular within the machine learning and programming language communities. The key idea of this new paradigm is to extend Turing-complete programming languages with probabilistic operations that include, for example, the drawing of (random) samples from a given probability distribution, the conditioning of random variables on observed outcomes, and the marginalization of random variables^{5,6}.

Such languages are sometimes referred to as *universal probabilistic programming languages* to clearly differentiate them from languages based on Bayesian networks, which have sometimes in recent years also been included in the probabilistic programming family. Here, we will use the terms “probabilistic programming” and “probabilistic programming language (PPL)” exclusively for universal languages.

Turing-completeness is an important concept in computer science, describing how expressive a programming language is. The famous Church-Turing thesis conjectures that any function, whose value can be computed by an algorithm, can be computed by a Turing-complete programming language. For instance, PPLs make it possible to use recursion (or loops) dependent on a stochastic expression when defining probabilistic models. This means that the graphical network describing the model is *dynamic* and can change during inference due to random sampling and observed data. In a PPL, the probabilistic model *is a program*, where the inference algorithm is not part of that program (the model). Hence, an alternative and potentially more intuitive name for a probabilistic program may be a *programmable model*: a model that is implemented as a program.

One of the earliest PPLs is Church⁷, which extends a functional subset of the Scheme programming language. Other PPLs (both universal and non-universal) include Figaro⁸ (a PPL embedded in Scala), WebPPL⁹ (a recent PPL embedded into JavaScript), Anglican¹⁰ (a general-purpose PPL embedded into Clojure that runs on the Java virtual machine), Venture¹¹ (a PPL with syntax similar to JavaScript), Edward¹² (a Python library for probabilistic modelling), Pyro¹³ (a PPL built on top of

*E-mail: fredrik.ronquist@nrm.se

†F.R., J.K., and V.S. contributed equally to this work.

^a<http://webppl.org>

^bFunctional programming (FP) is a programming paradigm, in which code is structured in units called *functions* that have no side effects; i.e. they only operate on a given input and produce an output but do not manipulate external objects.

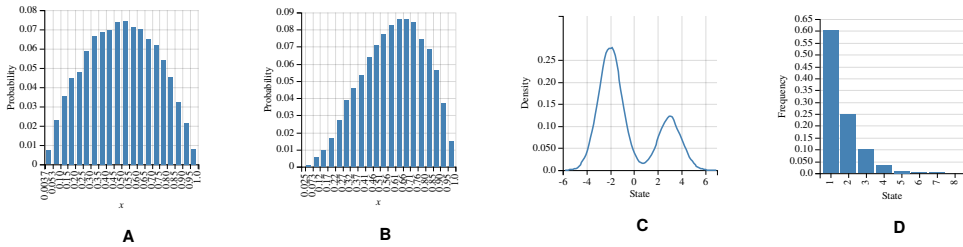


Fig. 1 **A** Prior distribution of the bias for the coin flip example. **B** Posterior distribution of the bias after observing heads once. **C** Mixture of two Gaussian models mixed using stochastic branching. **D** Resulting geometric distribution with $p = 0.6$. All plots are generated using the WebPPL environment.

PyTorch), Birch¹⁴ (a PPL that compiles into C++), and Stan¹⁵ (a platform for statistical modelling and computation). Note that this list is far from complete, and there exist many more experimental PPLs.

In this paper, WebPPL and Birch have been used for the reason of simplicity and efficiency, respectively. Some of the authors of this paper are currently developing a new domain-specific probabilistic programming language on top of the Miking¹⁶ platform. This language, called TreePPL^c, is designed specifically for the domain of statistical phylogenetics.

In the rest of this section, we describe the key concepts of probabilistic programming. The examples are given in WebPPL, but could easily be translated into any of the other universal PPLs. The WebPPL code can be run in a web browser through the WebPPL project web page^d. For a more comprehensive introduction to probabilistic programming, see for example the introductory text by van de Meent et al.¹⁷.

1.2 Sampling

The first key construct in probabilistic programs is *sample*, meaning that a value is drawn from a given probability distribution. Consider the following WebPPL code:

```
sample(Bernoulli({p: 0.5}))
```

The program models a simple coin flip scenario, where we sample from the Bernoulli distribution with probability 0.5, that is, a fair coin. When executed, the program returns either `true` or `false`, with probabilities corresponding to the sampled distribution.

Suppose we instead introduce another random variable x that models the probability of getting heads on the toss of the coin. Mathematically, such a model can be defined as follows:

$$x \sim \text{Beta}(\alpha, \beta)$$

$$y \sim \text{Bernoulli}(x)$$

where the beta distribution is used as a prior probability distribution for the value of x . The same model can be written as a PPL program (assuming we set $\alpha = \beta = 2$)

```
var x = sample(Beta({a: 2, b: 2}))
var y = sample(Bernoulli({p: x}))
```

Note that the `sample` construct is conceptually used to denote random variables, in this case the two variables x and y . If we run the program many times and plot the values of x , we get an approximation of the probability density function (PDF) for x in our mathematical model, that is, an approximation of the Beta(2, 2) distribution. Because the expected value of x is 0.5, y in the program still models an unbiased coin.

1.3 Conditioning and observations

Probabilistic programs are based on Bayesian statistics, and typically are intended to compute the posterior distribution, given a prior distribution and some observations. In the coin flip example, suppose we observe heads (encoded as `true`) after flipping the coin once. We want to infer $p(x|y)$, the posterior distribution of x , conditioned on the new observation $y = \text{true}$. As in the previous example, we assume that the prior distribution of x is Beta(2, 2). This model can be defined as follows.

```
var coinFlip = function() {
  var x = sample(Beta({a: 2, b: 2}))
  observe(Bernoulli({p: x}), true)
  return x
}
```

^c<https://treeppl.org/>

^d<http://webppl.org>

Note how the second `sample` construct is replaced with an `observe` construct. The program returns `x`, which is a (weighted) sample from the posterior distribution of x , computed by updating the prior distribution of x (defined explicitly in the model) with the observation that the coin flip resulted in `true` (conditioning on the observation). Supplementary Figure 1A shows the prior distribution of the bias x , which corresponds to the Beta(2, 2) distribution. Note how the posterior distribution in Supplementary Figure 1B has moved closer to 1 (bias towards heads), compared to the prior distribution.

The `observe` statement is a way of weighting a sample according to some distribution. It is basically equivalent to sampling from a distribution, followed by conditioning, using the `condition` statement covered in the main text^e. In WebPPL, there is also a construct called `factor`, which performs explicit weighting (also called scoring) of samples. A score or weight for a specific value can be computed using a distribution's PDF. In WebPPL, there is a `score` method that gives back the score of a value for a specific distribution. Thus, the `observe` statement in the previous example could be replaced with the following equivalent encoding using `factor`:

```
factor(Bernoulli({p: x}).score(true))
```

The `factor` construct is often used explicitly in the phylogenetic models described in this paper, especially in WebPPL.

1.4 Recursive models and stochastic branching

In the previous examples, the models were very simple. However, the power of universal probabilistic programming is that a model can be *any* program, which may include variable declarations and standard control flow operators. Consider the following program that includes an `if` statement:

```
var mixture = function(){
  if (sample(Bernoulli({p: 0.7}))) {
    return sample(Gaussian({mu: -2, sigma: 1}))
  } else {
    return sample(Gaussian({mu: 3, sigma: 1}))
  }
}
```

The model illustrates the use of *stochastic branching*, meaning that the paths taken in a program depend on the outcome of sampling. In the example, the guard of the `if` statement samples from the Bernoulli distribution. Depending on whether the `true` or `false` branch is taken, sampling of the resulting value is done with different Gaussian distributions (different μ values). The plot of the model is shown in Supplementary Figure 1C. As can be seen in the figure, the `true` branch has larger weight, because of the probability of 0.7 of it being chosen.

Stochastic branching can be combined with recursion: this is a key building block for phylogenetic models. Consider the following model, which describes a model of the geometric distribution:

```
var geometric = function(p) {
  if (sample(Bernoulli({p: p}))) {
    return 1
  } else {
    return geometric(p) + 1
  }
}
```

Note that there is no requirement of a deterministic termination of the recursion: the termination of the recursion depends on the stochastic branch, which each time depends on a different latent random variable. Supplementary Figure 1D shows the plot of `geometric(0.6)`. The simple recursion above generates a linear sequence of random length. We use similar recursions in our scripts to model the processes that generate bifurcating phylogenetic trees. We do that by including two recursive calls within the same function, one for each descendant of a speciation event.

1.5 Inference

The focus of this tutorial text has so far been on the model (the probabilistic program), and not on the inference algorithms. As discussed previously, in probabilistic programming, the choice of inference algorithm is intentionally separated from the model. For instance, using the `Infer` method of WebPPL, a user can apply the Sequential Monte Carlo (SMC) method to perform the inference of the coin example

```
Infer({model: coinFlip, method: 'SMC', particles: 20000})
```

or, alternatively, a Markov chain Monte Carlo (MCMC) method can be used:

```
Infer({model: coinFlip, method: 'MCMC', samples: 20000, burn: 5000})
```

The user also needs to specify the granularity of the approximation, using the number of particles or samples for SMC or MCMC, respectively.

In general, a key strength of the probabilistic programming paradigm is its expressive power, which is clearly shown in this paper within the domain of phylogenetics. One of the main research challenges within the PPL community is how to develop

^eAccording to the WebPPL documentation, for efficiency, the `observe` statement should be used instead of the combination of sampling and conditioning, especially for continuous distributions.

inference algorithms and compilers that scale to very large and complex models. Although this is an active area of research, our study hopefully demonstrates that already state-of-the-art PPL systems make it possible to perform effective inference on non-trivial phylogenetic models.

2 Tools for phylogenetic probabilistic programming

The paper is accompanied by a code repository containing all the sources, tools and data used for the study, including documentation. Specifically, the resources in the repository are designed to facilitate the use of two existing probabilistic programming languages—WebPPL and Birch—for phylogenetic inference. The code repository is available at:

<https://github.com/phypppl/probabilistic-programming>

The reader is referred to the `README.md` file in the repository, in which we describe how to install the tools and how to use them to rerun our analyses or to experiment with probabilistic programming for phylogenetic problems.

2.1 WebPPL for statistical phylogenetics

WebPPL is a universal probabilistic programming language based on JavaScript. We have written two packages, `phywppl` and `phyjs` that enable the reader to run phylogenetic simulations in WebPPL. The verification of the WebPPL programs relies on an auxiliary R package, `rppl`. Please refer to the aforementioned online documentation for further explanation.

2.2 Birch for statistical phylogenetics

Birch is a universal probabilistic programming language compiling into C++. The models presented in this paper are run like regular Birch packages. Refer to the aforementioned online documentation for further explanation.

2.3 Reading in phylogenetic trees

The WebPPL and Birch scripts we provide either simulate the diversification process along an observed reconstructed tree or computes the likelihood using analytical equations for such a tree. To facilitate the import of the observed tree data, we use a new JSON format for phylogenetic trees named PhyJSON¹⁸. Supported by the resources we provide in the repository, both WebPPL and Birch have mechanisms for reading in phylogenetic trees stored in the PhyJSON format. We also provide a stand-alone tool, `nexus2phyjson`, which can be used to convert trees in Nexus tree files to PhyJSON format¹⁸.

For convenience, we include several phylogenetic trees in the `phyjs` package for purposes such as testing and verification (Table 1). An up-to-date account of the included test trees is provided in the `webpppl/phyjs/README.md` file.

3 Diversification models

3.1 Basic notation and terminology

All of the diversification models considered in this study can be generically described as follows (see Table 2 for a summary of the notation). The process starts at some time $t_0 > 0$ in the past, where $t = 0$ represents the present time. Evolutionary lineages split (speciate) at a per-lineage rate λ and go extinct at per-lineage rate μ . The rates λ and μ are either constant, time-dependent or lineage-dependent, depending on the specific model. Each speciation event produces two lineages that further evolve independently of each other. The process is stopped when reaching the present ($t = 0$), at which point lineages still surviving are sampled (included in the observed tree) with probability $\rho \leq 1$.

The diversification process generates both trees with lineages that survive until the present, and trees that go completely extinct. Many surviving trees include side branches or whole subtrees that went extinct along the way. If the extinct parts and the branches leading to unsampled taxa are pruned away, we get what is called the “reconstructed tree”. In other words, the reconstructed tree is the subtree spanned by only those surviving lineages that have been sampled.

In diversification analyses, the focus is typically on reconstructed trees. For simplicity, we assume here that the reconstructed tree is known without error, but we note that it is straightforward to extend our probabilistic programming approach to accommodate uncertainty about the tree by drawing the tree from an appropriate tree sample. Learning the parameters of the diversification process involves computing the likelihood of one or more reconstructed trees given different parameter values.

Table 1 Example trees provided in the `phyjs` package.

Tree	Leaves	Age (Ma)	Description	Reference
<code>phyjs.bisse_32</code>	32	13.0	Example tree from Mesquite software	¹⁹
<code>phyjs.cetacean_87</code>	87	35.9	Cetacean tree from BMMtools package	²⁰
<code>phyjs.primates_233</code>	233	65.1	Primate tree from Diversitree package	²¹

Table 2 Summary of notation.

Symbol	Interpretation
λ	speciation (birth) rate
μ	extinction (death) rate
ϵ	turnover, μ/λ
ρ	probability of sampling a leaf
$\lambda(t)$	function characterizing time dependence of λ in some models
λ_0	initial λ , when λ varies over time
z	rate of exponential increase or decrease in λ
λ^i	speciation rate of process or branch i
$\lambda^i(t)$	time-dependent speciation rate of process i
μ^i	extinction rate of process i
z^i	exponential rate of increase or decrease in λ for process i
α	long-term trend in λ inheritance at speciation in ClaDS models
σ	standard deviation (log scale) in λ inheritance at speciation in ClaDS models
η	rate of switching of diversification process in the BAMB and LSBDS models
t_{MRCRA}	age of most recent common ancestor
ψ	reconstructed tree (extinct and unsampled side branches pruned away)
n	number of leaves in the reconstructed tree
V	the set of nodes (vertices) in the reconstructed tree
$a(i)$	index of immediate ancestor of node i
$l(i)$	index of left descendant of node i in oriented tree
$r(i)$	index of right descendant of node i in oriented tree
c	number of cherries (terminal bifurcations) in a tree
$P(\cdot)$	probability (density)
$L(\cdot)$	likelihood
$S(t, \theta)$	probability of process with parameters θ surviving from t until the present
Z	normalization constant of Bayes's theorem

We denote a reconstructed tree $\psi = (V, t)$, where V is a set of nodes (vertices) and t is a corresponding vector of speciation ages. The tree has n tips (terminal nodes or leaves) of degree one, $n - 1$ interior nodes of degree three, and the origin node of degree one. We index the nodes and their ages as follows:

- the origin has index 0;
- internal nodes have indices $\{1, 2, \dots, n - 1\}$, ordered in decreasing age;
- tips have indices $\{n, n + 1, \dots, 2n - 1\}$ (in any order)

The node V_1 corresponds to the first split between extant (surviving) lineages; it is referred to as *the most recent common ancestor* (MRCRA) or *the root* of the reconstructed tree. The age of a node i is t_i ; leaves have age 0. A subtree with root at node i and origin at time $t \geq t_i$ is denoted $\psi_i(t)$.

We will often find it convenient to distinguish between the two descendants of a node; without loss of generality, refer to them as the left and right descendant, respectively. A tree without leaf labels where nodes have been oriented in this way is an “oriented tree”²².

We define three mapping functions for indices in an oriented tree:

- $a(i)$ is the index of the immediate ancestor of node i
- $l(i)$ is the index of the left descendant of node i
- $r(i)$ is the index of the right descendant of node i

3.2 Conversions between tree spaces

In phylogenetics, we are interested in computing the likelihood of a labelled reconstructed tree, that is, a tree with leaf labels but with no distinction between the two descendants of a given ancestor. However, it is often convenient to derive the probability density of oriented trees without leaf labels first, and then convert it to a density on labelled trees without orientation²². The conversion factor is easy to find if we consider what happens if we start with a density on an oriented tree, then label it and finally remove the orientation. There are $n!$ unique ways of labelling an oriented tree, each with probability $1/n!$. When we remove the orientation, there are 2^{n-1} labelled oriented trees that produce the same labelled tree without orientation, where $n - 1$ is the number of interior nodes in the tree. Thus, the conversion factor is $2^{n-1}/n!$.

Table 3 Overview of phylogenetic diversification models considered in the paper.

Model	Full name	Reference
CRB	Constant rate birth model	Yule ²⁴ , Nee ²⁵
CRBD	Constant rate birth-death model	Feller ²⁶
TDB	Time-dependent birth model	Kendall ²⁷
TDBD	Time-dependent birth-death model	Kendall ²⁷
BAMM	Bayesian analysis of macro-evolutionary mixtures	Rabosky ²⁸
LSBDS	Lineage-specific birth-death shift model	Höhna et al. ²⁹
ClaDS[0-2]	Cladogenetic diversification rate shift models	Maliet et al. ³⁰

For completeness, we derive the conversion factor with the operations in the reverse order, first dropping the orientation and then applying the labels. When labels are missing, there are 2^{n-1-c} unique oriented trees for each tree without orientation, where c is the number of “cherries”. A cherry is a pair of leaves that are each other’s closest relatives²³; without labels the descendants of a cherry are identical and there is only one unique way in which they can be oriented. Labelling a tree without orientation is similarly affected by cherries, so that there are $n!2^{-c}$ unique label assignments. The conversion factor is thus $2^{n-1-c}/(n!2^{-c}) = 2^{n-1}/n!$.

In the literature on advanced diversification models, it is common practice to derive the density on unlabelled oriented trees and ignore the conversion to a density on labelled unoriented trees; in fact, the omission of this factor is rarely acknowledged. This contrasts with the derivation of the analytical likelihood for simple models, such as CRBD, where the conversion factor is almost always accommodated. Previous work on diversification models has focused on a single model and a single tree; in such cases, ignoring the conversion factor is not a problem. However, here we compare diversification models using Bayes factors, so the normalization constant needs to be computed consistently for all models, that is, based on the same outcome space.

For convenience, our simulations of diversification processes assume unlabeled and oriented trees. This makes the scripts simpler, and it facilitates comparison to previous descriptions of these models. Our simulations are weighted with the appropriate conversion factor to generate the density for labelled and unoriented trees. Thus, the normalization constants (marginal likelihoods) we compute are directly comparable to the likelihoods computed using the standard analytical equations established for the simple diversification models, such as CRBD²², for labelled and unoriented trees.

3.3 Conditioning on the age of the MRCA

A process that starts at some time t_0 in the remote past will produce a reconstructed tree that has a *stalk*, i.e., a branch leading to the MRCA. However, we usually do not have any information about the length of this stalk. For this reason, and others, it is often more convenient in practice to condition the process on the first split in the reconstructed tree, t_1 ²². This can easily be done by noting that t_1 can be considered the time of origin for both the left and the right subtrees originating from the first split, and that both of these lineages survived until the present by the very definition of the concept of MRCA. Thus, the probability of the reconstructed tree, conditioned on the age of the MRCA, is obtained by multiplying together the probabilities of the left and the right subtrees and by conditioning on their joint survival. Given an oriented tree ψ , now without the “stalk” from the origin to the most recent common ancestor, the likelihood is thus given by

$$L(\psi \mid \theta, t_1) = \frac{P(\psi_{l(1)}(t_1) \mid \theta, t_1)P(\psi_{r(1)}(t_1) \mid \theta, t_1)}{(S(t_1, \theta))^2}, \quad (1)$$

where θ is the vector of parameters of the model, and $S(t, \theta)$ is the probability of the process surviving (producing at least one sampled descendant) after time t .

3.4 Diversification models

In the paper, we consider nine different diversification models (Table 3).

The CRB(D) and TDB(D) models are simple diversification models, which assume that the process is the same for the entire tree, even though it can change over time. The other models (the advanced models) accommodate lineage-specific variation in diversification rates. The BAMM and LSBDS assume that the diversification process changes in a major way at certain points in time. In fact, the process is completely reset. Thus, LSBDS and BAMM can be described as models of punctuated change in diversification. The ClaDS models instead assume gradual, heritable changes in speciation and extinction rates. Specifically, this is modeled as small step-wise changes associated with speciation events (also called cladogenetic events).

We provide a tabular summary of the parameters of each model (Table 4) to facilitate comparison across them. We describe each model in detail below.

3.4.1 Constant rate birth-death models (CRBD and CRB)

The constant rate birth-death (CRBD) model is the simplest model considered here. Evolutionary lineages split at a constant per-lineage birth rate λ and go extinct at a constant per-lineage death rate μ . The parameter vector for this model is thus $\theta = (\lambda, \mu, \rho)$. As a special case, we consider the constant rate birth (CRB) model, also known as the Yule model, with $\mu = 0$ ²⁵.

Table 4 Summary of diversification model parameters.

Model	Parameters	Notes
CRB	λ	λ is speciation rate
CRBD	λ, ϵ	$\epsilon = \mu/\lambda$ is turnover rate
TDB	$\lambda(t), z$	z is exponential time-dependence parameter
TDBD	$\lambda(t), \epsilon, z$	
LSBDS	$\eta, \{(\lambda^i, \mu^i)\}$	η is change rate, i is index of process
BAMM	$\eta, \{(\lambda^i, \mu^i, z^i)\}$	z^i is time-dependence parameter of process i
ClaDS0	$\alpha, \sigma, \{\lambda^i\}$	α is trend parameter, σ is noise parameter in λ inheritance at speciation; i is branch index in complete tree
ClaDS1	$\alpha, \sigma, \mu, \{\lambda^i\}$	μ is extinction rate
ClaDS2	$\alpha, \sigma, \epsilon, \{\lambda^i\}$	ϵ is turnover rate

The probability that a CRBD process starting at time t survives until the present and is sampled is known analytically²²; it is

$$S(t, \lambda, \mu) = \frac{r}{\lambda - (\lambda - r/\rho) e^{-rt}}, \quad (2)$$

where $r = \lambda - \mu$ is known as the “net diversification rate”. The likelihood of a reconstructed tree conditioned on the time of the MRCA is also known analytically; it is given by:

$$L(\psi|\theta, t_1) = \frac{2^{n-1}}{n!} \lambda^{n-2} \rho^n \frac{\widehat{g}(t_1)^2 \prod_{i=2}^{n-1} \widehat{g}(t_i)}{\widehat{g}(0)^n S(t_1)^2}, \quad (3)$$

where

$$\widehat{g}(t) = \frac{e^{-rt}}{(\lambda - (\lambda - r/\rho) e^{-rt})^2}. \quad (4)$$

Even though the likelihood is known analytically, there are no conjugate priors for λ and μ that would yield an analytical posterior. Thus, we end up with an intractable integral if we want to learn these parameters from one or more observed trees.

3.4.2 Time-dependent birth-death models (TDBD and TDB)

In the time-dependent birth-death model (TDBD), the speciation rate is assumed to change continuously through time. More specifically, we consider the following time-dependence:

$$\lambda(t) = \lambda_0 e^{z(t_1-t)}. \quad (5)$$

Thus, λ_0 is the speciation rate prevailing at the time of the MRCA. Furthermore, if $z < 0$ (resp. $z > 0$), the speciation rate decreases (resp. increases) exponentially when going toward the present. An exponentially decreasing speciation rate can be seen as an approximate model for diversity dependence. The parameter vector for this model is $\theta = (\lambda_0, \mu_0, x, \rho)$.

The likelihood appears to be intractable for the present model with exponentially varying speciation rate and constant extinction rate. On the other hand, a simple solution is available for the slightly different model examined here, in which λ and μ are both exponentially decreasing or increasing at the same rate z (and thus the turnover rate λ/μ is constant):

$$\begin{aligned} \lambda(t) &= \lambda_0 e^{z(t_1-t)}, \\ \mu(t) &= \mu_0 e^{z(t_1-t)}. \end{aligned}$$

Under this model, the probability that a lineage starting at time t survives until the present and is sampled is now (for a general method of deriving the likelihood for time-dependent birth-death models, see Morlon et al.³¹):

$$S(t, \lambda_0, \mu_0, z) = \frac{r_0}{\lambda_0 - (\lambda_0 - r_0/\rho) e^{-(r_0/z)(1-e^{-zt})}}, \quad (6)$$

where $r_0 = \lambda_0 - \mu_0$. The likelihood of a reconstructed tree conditioned on the time of the MRCA has the same general form as for the CRBD:

$$L(\psi|\theta, t_1) = \frac{2^{n-1}}{n!} \rho^n \frac{\widehat{g}(t_1)^2 \prod_{i=2}^{n-1} \widehat{g}(t_i) \lambda(t_i)}{\widehat{g}(0)^n S(t_1)^2} \quad (7)$$

with S such as just given and:

$$\widehat{g}(t) = \frac{e^{-(r_0/z)(1-e^{-zt})}}{(\lambda_0 - (\lambda_0 - r_0/\rho) e^{-(r_0/z)(1-e^{-zt})})^2}. \quad (8)$$

As a special case, we consider the time-dependent birth (TDB) model, which is equivalent to TDBD except that there is no extinction, that is, $\mu = 0$. Note that the TDBD model collapses to CRBD when $z = 0$. Similarly, TDB becomes equivalent to CRB when $z = 0$.

3.4.3 Bayesian analysis of macroevolutionary mixtures (BAMM)

The BAMM model was proposed by Rabosky²⁸. The original formulation of the change process is statistically incoherent³² but it is straightforward to fix this, and we follow the slight reinterpretation of the model suggested by Moore et al.³². In this version, BAMM is an episodic, Poisson-modulated, birth-death process with exponentially decaying speciation rate. To describe the process, consider a generic lineage e at time t . At this time point, the lineage is associated with rate parameters with index $e(t)$. Specifically, the lineage carries with it a triplet of rates $(\lambda^e(t), \mu^e(t), z^e(t))$. Then:

- at rate $\mu^e(t)$, the lineage goes extinct;
- at rate $\lambda^e(t)$, the lineage splits into two lineages (say f and g), in which case the two daughter lineages inherit the current rates, i.e.

$$\begin{aligned}(\lambda^f(t), \mu^f(t), z^f(t)) &= (\lambda^e(t), \mu^e(t), z^e(t)), \\ (\lambda^g(t), \mu^g(t), z^g(t)) &= (\lambda^e(t), \mu^e(t), z^e(t));\end{aligned}$$

- $\lambda^e(t)$ increases or decays exponentially at rate z ;
- at rate η , the triplet of rate parameters is redrawn from a pre-specified trivariate distribution Φ , i.e.

$$(\lambda^e(t_-), \mu^e(t_-), z^e(t_-)) \sim \Phi \quad (9)$$

The process starts with a single lineage a at some time t_0 , with rate parameters $(\lambda^a(t_0), \mu^a(t_0), z^a(t_0)) \sim \Phi$. Specifically, we start the process at the time immediately before the first split in the tree (at the MRCA), and we assume that the process index at this point is $a(\text{MRCA}) = o$ (o for origin). The prior distributions used in this paper for Φ were chosen to harmonize with the priors used for other models, as specified in the section on priors below. The process stops when reaching the present ($t = 0$) and the surviving lineages are sampled with probability ρ .

The likelihood under the BAMM model as defined here does not have an analytical solution, nor does it seem to be amenable to any known numerical techniques for solving the complex differential equations involved³². A recent paper³³ analyzes a variant of the BAMM model, which differs from the variant considered here in that the rate of diversification model shifts is considered to be constant for a clade, regardless of the number of lineages the clade contains. This is a somewhat unusual type of model, as lineages are usually considered to evolve independently and not as parts of a larger clade. Nevertheless, under this assumption they succeed in deriving an analytical expression of the likelihood for the case of a single shift. They then extend this to multiple shifts, keeping the likelihood computations manageable by assuming that a clade that has shifted diversification rates once cannot shift again.

Describing the BAMM model as defined here (or in the clade-spanning variant) using a PPL is straightforward. Effective inference can then be performed using generic techniques available for PPLs, such as SMC or PMCMC (particle Markov chain Monte Carlo), as we show in the current paper. No artificial model constraints need to be introduced.

3.4.4 LSBDS

The recently proposed LSBDS model²⁹ can be seen as a specialized version of the BAMM model, in which $z = 0$ at all times and for all lineages. In other words, there is no exponential decay of speciation rates; the speciation rate remains constant between rate shift events. As a result, Φ is now a bivariate distribution.

Under these conditions, it becomes possible to compute the likelihood of a reconstructed tree by approximating Φ as a product of two discrete distributions, with a finite (and small) number of possible values for λ and μ , and then relying on standard numerical techniques for solving the differential equations involved^{34,29}. This is similar to the discretization approach frequently used in phylogenetics in order to efficiently approximate the likelihood when rates vary across sites according to a continuous gamma distribution³⁵.

Using this approach, the backward-in-time recursion for the extinction probability and for the conditional likelihood, which are both conditioned on the current value of λ and μ for the lineage under consideration, entails a set of LM coupled master equations, where L and M are the number of bins used for the λ and μ distributions, respectively. In practice, this imposes a rather strict constraint on the number of discretization bins that can be used, as the computational complexity otherwise becomes unmanageable. We note that the empirical examples discussed in the LSBDS paper all use a fixed value for μ across the tree, thus effectively setting $L = 1$. The SMC techniques we use in the current paper do not suffer from such limitations, as they rely on sampling values from the λ and μ distributions.

Another model that also eliminates the z variable of the BAMM model was presented recently³⁶. This model is based on restricting the number of possible diversification models to a finite number k of different diversification model categories. An MCMC algorithm is then used to sample over different histories of shifts among these categories over the tree. To simplify the computation of likelihoods, the authors further assume that no shifts among model categories occur on extinct side branches. In our PPL and SMC context, we do not gain much by restricting the number of diversification rate models, and there is no need to simplify the treatment of extinct side branches. Thus, we do not consider this model further here.

3.4.5 The cladogenetic diversification rate shift models (ClADS)

The ClADS models³⁰ assume that the speciation rate changes by a small random amount at each speciation event. The extinction rate is assumed to be either equal to 0 (ClADS0), constant but positive (ClADS1), or proportional to the speciation rate, such that the turnover rate ($\epsilon = \mu/\lambda$) is constant (ClADS2). Thus, in all cases, $\mu = \mu(\lambda)$ can be seen as a (possibly constant, for ClADS0 and ClADS1) deterministic function of λ .

Consider a generic lineage e at time t . This lineage carries with it a rate λ^e . Then:

- at rate $\mu(\lambda^e)$, the lineage goes extinct;
- at rate λ^e , the lineage splits into two lineages (say f and g), in which case the two daughter lineages draw their respective speciation rates, λ^f and λ^g as follows:

$$\log \lambda^f \sim \mathcal{N}(\log(\alpha\lambda^e), \sigma^2).$$

$$\log \lambda^g \sim \mathcal{N}(\log(\alpha\lambda^e), \sigma^2).$$

The process starts with a single lineage o at some time t_0 , with speciation rate λ^o .

The α parameter introduces a deterministic long-term trend in the otherwise random variation of λ through time, across the many speciation events typically occurring over the complete phylogeny. When $\alpha < 1$ (resp. $\alpha > 1$), the logarithm of the speciation rate decreases (resp. increases) on average at a speciation event.

The original ClADS paper³⁰ focuses on the mean diversification rate multiplier $m = \alpha \exp(\sigma^2/2)$ rather than on α . However, we prefer the α parameterization because it allows us to find a convenient conjugate prior that supports delayed sampling and thus makes SMC inference more efficient (see below).

There is some superficial resemblance between $\log \alpha$ and the z parameter in the TDBD, TDB and BAMM models, also suggesting that this would be a natural parameterization. However, the dynamics of the ClADS models is quite different from the TDBD, TDB and BAMM models. This makes the interpretation of both the α and m parameters more complicated than what is immediately obvious. When there is noise in the ClADS models ($\sigma > 0$), there will be variation across lineages in diversification rate. This will result in lineage selection, ensuring that the average diversification rate across lineages goes up more than suggested by the values of m or α ³⁷. Even within lineages, there will be slight deviations from the behavior that might be expected from the m (or α) values. For instance, when there is noise, setting $m = 1$ will result in the expected diversification rate at the end of some specified time period being lower than the starting rate. This is because the process is more likely to reach the end of the time period without any further change if the diversification rate goes down than if it goes up.

We note that it would be straightforward to estimate posterior distributions on m using our scripts even though they use the α parameterization. This can be achieved either by adding a line that computes m from α and σ and then returns it to the model scripts before running the analysis, or by converting the sampled α and σ values to m values after the analysis has completed.

The likelihood under the ClADS1 and ClADS2 models is not analytically available, but it can be numerically evaluated³⁰. The evaluation method originally developed for the ClADS models involved various numerical approximation techniques, including discretization of time and rate space, and expands over thousands of lines of code in the RPANDA R package. Recently, considerably more efficient inference techniques based on data augmentation have been developed for these models³⁷.

4 Prior probability distributions

To facilitate the interpretation of the Bayes factor tests, we standardized prior probability distributions across diversification models as much as possible in our analyses. Before going into details, it may be helpful to explicitly declare the parameterizations we assume for the statistical distributions used. Thus, for the exponential distribution, we assume the rate parameterization, for the inverse gamma distribution we use the shape-scale parameterization, and finally, for the normal (Gaussian) distribution, the parameters are the mean and the variance of the distribution.

Across all models, we used an Exponential(1) prior for the speciation rate, and a Uniform(0, 1) prior for the turnover rate, both common priors in the diversification model literature. The specific implementations are listed for each model below. For the σ parameter of the ClADS models, Maliet et al.³⁰ used a prior with most probability mass close to 0 ($\sigma \sim \text{InvGamma}(1, \log(1.1))$). Upon examination of the empirical results published in the same paper (shown in their Figure 4a), we concluded that this choice is overly conservative. That is, the prior puts so much probability on low values of σ that the posterior may underestimate the extent of lineage-specific variation in diversification rates. We also note that it is more natural to consider an inverse gamma prior for the variance rather than the standard deviation of the normal distribution, since this is a conjugate prior for the normal distribution. Therefore, we used a $\sigma^2 \sim \text{InvGamma}(1, 0.2)$ prior in our analyses.

The original ClADS paper³⁰ used an improper prior for the α parameter. This is not suitable for our purposes, as we need to simulate from the prior in SMC. We instead assumed $\log \alpha \sim \mathcal{N}(0, \sigma^2)$. By making the variance of the $\log \alpha$ prior dependent on σ^2 , we establish a conjugate normal-inverse-gamma prior. This results in a joint prior on $(\log \alpha, \sigma^2)$ that has its mode for α at 0, at which point there is neither acceleration nor deceleration of speciation rates. The posterior distribution of α values reported earlier for the bird trees under the Clads2 model³⁰ is also well covered by this joint prior. For z , we used the prior proposed in the original BAMM paper²⁸, namely $z \sim \mathcal{N}(0, 0.05^2)$.

Finally, for the LSBDS and BAMM models, we wanted a prior on η that was scaled to time. In the LSBDS and BAMM papers^{29,28,38}, it has been common to instead specify a prior scaled to the total length of the tree. This allows one, for instance, to

specify a prior with an expectation of one change in the diversification process over the reconstructed tree. However, if the changes we observe in diversification rates are the result of some evolutionary process, then it would seem more reasonable to assume that the expected number of changes is a function of evolutionary time rather than of an arbitrarily circumscribed reconstructed tree. To obtain this effect, while still maintaining some scaling to tree size, we chose a prior with one expected change in diversification rates over the time period from the most recent common ancestor to the present. For a small reconstructed tree, this would correspond to an expectation of slightly more than one change over the tree, while the expectation could be more than a few changes in a big tree. Specifically, we assumed $\eta \sim \text{Exponential}(t_{\text{MRCNA}})$, where t_{MRCNA} is the age of the first split in the tree.

For completeness, all prior probability distributions are listed below for each of the examined models (see also Supplementary Figure 2).

4.1 CRB

The CRB model has only one parameter, λ , for which we use the standard prior:

$$\lambda \sim \text{Exponential}(1).$$

4.2 CRBD

The CRBD model has two parameters, λ and μ . For λ we use the standard prior, and for μ the indirect prior induced by assuming a uniform prior on the turnover rate $\epsilon = \mu/\lambda$.

$$\begin{aligned} \lambda &\sim \text{Exponential}(1), \\ \epsilon &\sim \text{Uniform}(0, 1). \end{aligned}$$

4.3 TDB

For the TDB model, we applied the standard priors as follows:

$$\begin{aligned} \lambda_0 &\sim \text{Exponential}(1), \\ z &\sim \mathcal{N}(0, 0.05^2), \end{aligned}$$

where λ_0 is the initial speciation rate, and z is the time dependence parameter in $\lambda(t) = \lambda_0 e^{zt}$. In other words, the standard λ prior applies to the initial speciation rate in this model.

4.4 TDBD

The TDB priors are extended to the TDBD case as follows:

$$\begin{aligned} \lambda_0 &\sim \text{Exponential}(1), \\ z &\sim \mathcal{N}(0, 0.05^2), \\ \epsilon &\sim \text{Uniform}(0, 1), \end{aligned}$$

where λ_0 is the initial speciation rate, z is the time dependence parameter in $\lambda(t) = \lambda_0 e^{zt}$, and ϵ is the turnover rate. Note that in our implementation we have kept the turnover rate constant (rather than the extinction rate), i.e., $\mu(t) = \epsilon\lambda(t)$.

4.5 ClaDS0

For the ClaDS0 model, we applied the standard λ prior to the initial speciation rate, in line with the TDB(D) models. The α and σ priors are justified above. Specifically, the ClaDS0 priors we used are:

$$\begin{aligned} \lambda_0 &\sim \text{Exponential}(1), \\ \sigma^2 &\sim \text{InvGamma}(1, 0.2), \\ \log \alpha &\sim \mathcal{N}(0, \sigma^2), \end{aligned}$$

where λ_0 is the initial speciation rate, σ^2 represents the variance in the inherited speciation rate and α is the *speciation trend parameter*.

4.6 ClaDS1

The prior distributions related to the speciation rates are the same as for ClaDS0. In addition, we assume

$$\epsilon \sim \text{Uniform}(0, 1),$$

where ϵ is the initial turnover rate. The extinction rate, $\mu = \epsilon\lambda_0$, remains constant in the whole tree.

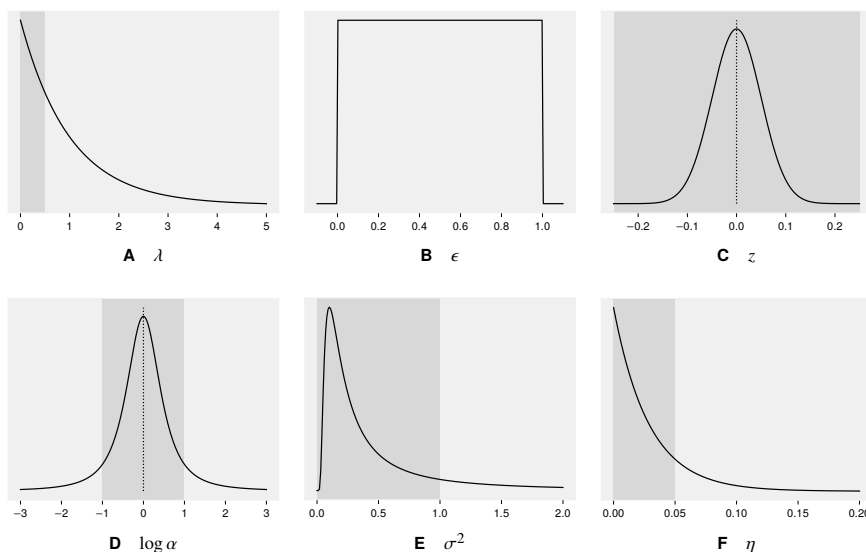


Fig. 2 Prior distributions of model parameters. The shaded regions correspond to the region of parameter space illustrated in the posterior plots for the empirical analyses (Supplementary Figures 13–22). See also Supplementary Figure 23.

4.7 ClaDS2

The prior distributions related to the speciation rates are the same as for ClaDS0. In addition, we assume

$$\epsilon \sim \text{Uniform}(0, 1),$$

where ϵ is the turnover rate. Unlike ClaDS1, the extinction rate changes at each speciation such that the turnover remains constant over the whole tree.

4.8 LSBDS

For the LSBDS model, we define the joint prior Φ , such that the all λ^i values, including the speciation rate for the MRCA (λ^0), are drawn from the standard λ prior used for other models, and such that the μ^i values, including the value of the MRCA, are drawn independently from the distribution induced by drawing ϵ from the standard uniform distribution used for other models. Specifically,

$$\eta \sim \text{Exponential}(t_{\text{MRCA}}),$$

$$\lambda^i \sim \text{Exponential}(1),$$

$$\epsilon^i \sim \text{Uniform}(0, 1),$$

where η is the rate of shifts in diversification processes, t_{MRCA} is the time (age) of the most recent common ancestor, and λ^i and ϵ^i are the speciation and the turnover rates of the i -th diversification process.

BAMM

The prior distributions for BAMM are the same as for LSBDS, with addition of

$$z^i \sim \mathcal{N}(0, 0.05^2),$$

where z^i is the time dependence parameter for the speciation rate of the i -th diversification process. This is the same prior distribution used for the z parameter of the TDB and TDBD models.

5 PPL model descriptions

In this section, we describe the PPL model scripts we used in the paper. We focus on WebPPL, as we think these model scripts are the most accessible to biologists. We start by describing model scripts that make use of the analytical likelihood equations.

We then present the complete description of the explicit simulation script for the CRBD model that is partly covered in the main paper. Finally, we provide a brief overview of the simulation scripts for the remaining models. We end the section with a brief discussion of how the Birch scripts are similar to and how they differ from the WebPPL scripts. For full details, we refer the interested reader to the code repository accompanying the paper.

5.1 Scripts based on analytical likelihoods

As mentioned in Section 3, the likelihood of a reconstructed tree conditioned on the age of the MRCA and the parameters of the diversification process is known analytically for the simple diversification models (CRB, CRBD, TDB, TDBD). We can take advantage of this in probabilistic programs, facilitating efficient inference of model parameters, by scoring simulations according to the analytical likelihood. To simplify the implementation of such scripts, we provide the analytical likelihoods as deterministic functions in the `phyjs` library. Four functions are available. The function `exactCRBDLikelihoodComplete` (`tree`, `lambda`, `mu`) computes the likelihood of a reconstructed tree under the CRBD model for specific values of λ and μ , assuming complete sampling of the leaves (tips) in the tree, $\rho = 1$. The function `exactCRBDLikelihoodRandom` (`tree`, `lambda`, `mu`, `rho`) computes the same likelihood when the leaves are randomly sampled with probability $\rho < 1$. Finally, the functions `exactTDBDLikelihoodComplete` (`tree`, `lambda`, `mu`, `z`) and `exactTDBDLikelihoodRandom` (`tree`, `lambda`, `mu`, `z`, `rho`) compute the corresponding probabilities for the TDBD model. By setting `mu = 0`, the functions can be used to compute the likelihoods for the CRB and TDB models.

The following listing shows how to infer the posterior distribution of λ and ϵ for the CRBD model using the analytical likelihood and the MCMC inference method:

Algorithm 1 CRBD model with analytical likelihood.

```

1 var tree = phyjs.bisse_32
2
3 var model = function() {
4   var lambda = exponential({ a: 1 })
5   var epsilon = uniform({ a:0.0, b: 1.0 })
6   var mu = epsilon*lambda
7
8   factor( exactCRBDLikelihoodComplete(tree, lambda, mu) )
9
10  return [lambda, epsilon]
11 }
12
13 var dist = Infer({method: 'MCMC', samples: 100, lag: 10, burn: 1000, model: model})
14
15 dist

```

In the script, we first select one of the provided trees in the `phyjs` package. The model is then set up by specifying the priors on the model parameters, and computing the value of the extinction rate μ , encoded as the variable `mu`. The simulation then scores the simulation according to the analytical likelihood of the sampled parameter values using the `factor` construct. In the final line of the model function, the values of the model parameters are returned.

For inferring the posterior distribution induced by the model function, the MCMC method is a good choice. For explanation of the inference settings, see the WebPPL documentation of the MCMC method^f. The last line ensures that the estimated joint posterior distribution, encoded as `dist`, is printed.

The script can be run using the commands we provide in the code repository accompanying this paper^g, as explained in the documentation provided there. In the directory `webppl/phywppl/examples/` in the repository, we provide analytical scripts of this kind for the CRB, CRBD, TDB and TDBD models.

5.2 Basic script for CRBD

Here, we give a complete WebPPL implementation of the CRBD model. The program describing the model is divided into two files to facilitate reuse of the code. The simulation part is specified in one file, and the analysis part in another. The simulation file contains code that simulates the CRBD process along a given tree for specified values of the model parameters. This file can be reused unaltered regardless of the particular analysis one wants to perform. The analysis file contains the specification of the priors, the data, and the inference method. This file needs to change from one analysis to another.

The analysis file (Algorithm 2) is structured in the same way as the script using the analytical likelihood for CRBD. However, instead of calling a function to compute the analytical likelihood, we call the simulation function for the CRBD model. This function weights the simulation appropriately for the given parameter values, conditioned on the observed reconstructed tree. The model function returns the model parameters, as before. However, instead of inferring the posterior distribution on those parameters, we now use SMC and focus on the normalization constant (the marginal likelihood or model evidence). The normalization constant estimate is available in the `normalizationConstant` property of the distribution object returned by the

^f<http://docs.webppl.org/en/master/inference/methods.html#mcmc>

^g<https://github.com/phywppl/probabilistic-programming>

Infer function when the method is SMC. Similar example scripts are available in the `webppl/phywppl/examples/` directory for all models studied in the paper.

Algorithm 2 Analysis script for CRBD simulation.

```

1 var tree = phyjs.read_phyjson("bisse_32.phyjson")
2
3 var model = function() {
4   var lambda = exponential({ a: 1 })
5   var epsilon = uniform({ a:0.0, b:1.0 })
6   var mu = epsilon*lambda
7
8   simCRBDNaive( tree, lambda, mu)
9
10  return [lambda, epsilon]
11 }
12
13 var dist = Infer({method: 'SMC', particles: 10000, model: model})
14
15 dist.normalizationConstant

```

Let us now turn to the simulation script (Algorithm 3). The script presented here is a naive PPL implementation of the CRBD model in that it does not use the analytical likelihood. Instead, it explicitly simulates the speciation and extinction process conditioned on the reconstructed tree. The script is also naive in the sense that it does not include any modifications to support aligned SMC inference, which is important for improving inference efficiency. The advanced inference techniques we used in the paper, including alignment, are discussed in Section 6. The script forms a basic template that can be used to express all diversification models analyzed in our paper. It should also be straightforward to extend the script to a range of new diversification models that have not been explored previously.

Algorithm 3 A complete WebPPL script for simulating CRBD.

```

1 var goesExtinct = function( startTime, lambda, mu )
2 {
3   var t = exponential( {a: lambda + mu} );
4
5   var currentTime = startTime - t;
6
7   if ( currentTime < 0 ) {
8     return false
9   }
10
11  var speciation = flip( lambda/(lambda+mu) )
12  if ( !speciation )
13    return true;
14
15  return( crbdGoesExtinct( currentTime, lambda, mu )
16         && crbdGoesExtinct( currentTime, lambda, mu ) );
17 }
18
19 var simBranch = function( startTime, stopTime, lambda, mu )
20 {
21  var t = exponential ( {a: lambda} );
22
23  var currentTime = startTime - t;
24
25  if ( currentTime <= stopTime )
26    return 0.0;
27
28  factor( Math.log( 2.0 ) );
29  condition ( crbdGoesExtinct( currentTime, lambda, mu ) )
30
31  return simBranch( currentTime, stopTime, lambda, mu )
32 }
33
34 var simTree = function( tree, parent, lambda, mu )
35 {
36  factor( - mu * ( parent.age - tree.age ) );
37
38  simBranch( parent.age, tree.age, lambda, mu );
39
40  if ( tree.type == 'node' )
41  {
42    factor( Math.log( lambda ) );
43
44    simTree( tree.left, tree, lambda, mu )

```

```

45     simTree( tree.right, tree, lambda, mu )
46   }
47 }
48 }
49
50 var simCRBDNaive = function( tree, lambda, mu )
51 {
52   var numLeaves = phyjs.countLeaves( tree )
53   var corrFactor = ( numLeaves - 1 ) * Math.log( 2.0 ) - phyjs.lnFactorial( numLeaves )
54   factor( corrFactor )
55
56   simTree( tree.left, tree, lambda, mu )
57   simTree( tree.right, tree, lambda, mu )
58 }

```

The main function in the script is `simCRBDNaive`, defined at the end of the script. It takes three parameters: the model parameters `lambda` and `mu`, and the `tree` on which to condition the simulation (note the actual implementation order). For simplicity, the process is simulated along an oriented and unlabelled tree (see Section 3.2). This assumption allows us to ignore the probability factor associated with rotation and labeling of the reconstructed tree during the main part of the simulation. To ensure that the simulation nevertheless carries the right weight, it is first endowed with the appropriate rotation and labeling probability (see Section 3.2) using two utility functions in the `phyjs` library and the `factor` construct in WebPPL. This is important for computing the correct normalizing constant, but does not affect inference otherwise, since this probability factor is the same for all simulations. Note that, for numerical stability, the particle weights in WebPPL are stored as logarithms.

Next, the function `simTree` is called on both children of the root node (the MRCA), initiating the recursion over the observed tree. Note that `simCRBDNaive` does not return anything. It is called only for the side-effect of weighting the sampled `lambda` and `mu` values by conditioning the simulation on the observed tree.

The function `simTree` is similar in structure to `simCRBDNaive`: it computes various weights and, if we have not reached a leaf, continues the recursion. Here, we present a naive implementation of `simTree`, where the execution of the probabilistic program is reweighted as soon as the information becomes available via calls to `factor`. A bit later, when we discuss advanced inference techniques (Section 6), we will show a version of the script, which only reweights the particle after the hidden side-branches have been processed. In the naive version, the first step is to factor the probability of no extinction on the branch from the parent to the node (line 36). This corresponds to observing zero extinction events from a Poisson distribution parameterized by the extinction rate $|\mu|$ and the branch length $|\text{parent.age} - \text{tree.age}|$, i.e. the weight can be obtained by plugging in 0 in the probability density function (pdf) of the Poisson distribution. Next, we simulate the hidden side-branches (line 38), the function will re-weight the computation accordingly. Finally, if we are at a node, we observe an immediate speciation event, i.e. 0 from an exponential distribution with parameter $|\text{lambda}|$, which has the weight of $|\text{Math.log}(\text{lambda})|$.

The `simBranch` function recursively simulates speciation events along the branch. If there is a speciation event, the side branch it generates must go extinct, as it is not present in the observed reconstructed tree. We call such a speciation event a “hidden” speciation because it is not visible in the observed tree. To condition the simulation on the extinction of the side branch resulting from a hidden speciation, we require the call to the recursive simulation function `goesExtinct` to return `true`. The `goesExtinct` function is described in the main paper; it is defined at the top of the script presented here. It simulates an outcome of the birth-death process for given `lambda` and `mu` values, starting at a given time in the past and counting downwards until the present (time 0). If all lineages go extinct before reaching the present, the function returns `true`, otherwise it returns `false`. In connection with the call to `goesExtinct`, the `simBranch` function also needs to take a rotational factor into account. This arises because there are two indistinguishable simulations that correctly account for the tree we condition on: one in which the right descendant of the hidden speciation event goes extinct and the left descendant gives rise to the observed continuation of the lineage, and one in which the left descendant goes extinct and the right descendant gives rise to the observed continuation of the lineage. Thus, the correct probability score for the simulation is twice what would have resulted from a single call to `goesExtinct`, and we therefore need to add $\log 2$ to the weight (recall that probability factors are represented on the log scale in WebPPL) before continuing the recursion.

The analysis and simulation scripts described above are simplified versions of the example script `crbd-naive.wpp1` in the `webppl/phywpp1/examples/` directory, and the similarly named model script in the `webppl/phywpp1/models/` directory. The simulation script presented here differs in four details from the model script in the repository. First, the script in the repository accommodates the possibility of incomplete sampling of the leaves in the tree. Thus, there is an additional parameter ρ in the model, encoded as the variable `rho` in the script. This variable appears as an argument to all simulation functions. The `goesExtinct` function needs to take the sampling probability into account, and is aptly renamed to `goesUndetected`.

Second, the definitions of the `simTree`, `simBranch` and `goesUndetected` functions are hidden inside the `simCRBDNaive` function. This allows us to use the same generic names for these functions in all diversification models; only the simulation function needs to have a unique name. Hopefully, this facilitates for readers to recognize how we extended the basic template to accommodate the other diversification models.

Third, the script in the repository employs guards against extreme values of the `lambda` variable, which can otherwise cause problems with numerical exceptions or stalled simulations. We solve these problems by assigning zero weight to the simulation if the `lambda` value is above or below certain threshold values. We verified that the discarded simulations have negligible impact on the inference for all the examined models using the chosen guard values.

Finally, unlike the simple script described here, the script in the repository corrects for survivorship bias as explained in the

next section. Before moving on to this, we want to point out that the naive CRBD simulation is suitable mainly for exploratory analyses of small trees. For efficient inference in WebPPL on phylogenetic diversification models for larger trees, it is important to manually modify the scripts so that they support aligned SMC inference (see Section 6.1). The CRBD model is the only model for which we provide an unaligned (“naive”) model script.

5.3 Correcting for survivorship bias

As discussed above (Section 3.3), if we condition the simulation on the age of the MRCA, we implicitly condition on the survival of the two subtrees originating at this point in time. To do this in a probabilistic program, we need to divide the probability of a simulation by $S(t_{\text{MRCA}}, \theta)^2$, that is, the square of the probability that the process survives (and is sampled) if it starts at t_{MRCA} , and the model parameter values are θ . If $S(t, \theta)$ is not available in closed form, this is potentially cumbersome because it involves a sum and integral over an infinite number of realizations of the process for each simulation. However, we can solve this by observing that the division by $S(t_{\text{MRCA}}, \theta)^2$, which we cannot evaluate in general, can be rewritten as follows:

$$p(\theta|\psi, \text{survival}) \propto \frac{p(\theta)p(\psi|\theta)}{S(t_{\text{MRCA}}, \theta)^2} = p(\theta)p(\psi|\theta) \sum_{M=1}^{\infty} M (1 - S(t_{\text{MRCA}}, \theta)^2)^{M-1} S(t_{\text{MRCA}}, \theta)^2. \quad (10)$$

This shows that we can correct for the survivorship bias by using the generative model encoded in the function `goesExtinct` (or `goesUndetected`) to simulate two evolutionary processes starting at t_{MRCA} . We repeat this until both simulations survive to the present time, and multiply the weight of the rest of the simulated diversification process along the observed tree by the number of repetitions required to achieve this.

In WebPPL, we use the following recursive function to compute the number of simulations required until both trees survive:

```
var M_goesExtinct = function( t, lambda, mu )
{
  if ( !goesExtinct( t, lambda, mu ) && !goesExtinct( t, lambda, mu ) )
    return 1
  else
    return 1 + M_goesExtinct( t, lambda, mu )
}
```

The following lines are then inserted at the end of the simulation function `simCRBDNaive` to correctly condition on the survival of the two subtrees defining the MRCA:

```
var M = M_goesExtinct( t, lambda, mu )
factor( Math.log( M ) )
```

The script in the repository is slightly more complex because we take incomplete sampling into account, and also implement a guard against an excessive number of repetitions.

5.4 Scripts for other diversification models

Example analysis scripts for all models are provided in the directory `webppl/phywpp1/examples/`, and generic model simulation scripts in the directory `webppl/phywpp1/models/`. All simulation scripts we provide in the latter directory are set up to trigger aligned SMC inference in WebPPL. As mentioned above, the only exception is the CRBD model, where we provide both a naive, unaligned version (`phywpp1/models/crbd-naive.wpp1`) and an aligned version (`phywpp1/models/crbd.wpp1`) for instructional and testing purposes. We provide both scripts using analytical likelihoods and scripts using explicit simulation for all simple diversification models (CRB, CRBD, TDB, TDBD). The scripts for the CRB, TDB and TDBD models involve simple and straightforward modifications of the corresponding scripts for the CRBD model, described above.

The model scripts for all lineage-specific diversification models (ClaDS0, ClaDS1, ClaDS2, LSBDS, BAMM) follow the template described above for the CRBD model, including the modifications needed to trigger aligned SMC inference. Analogous component functions are used in the simulation scripts; they are even named the same except for the main simulation function, which is named after the corresponding diversification model.

In probabilistic programming, you have to be explicit about the model variables that you want to estimate. These are the variables that are returned from the model function. The focus in our study was on the normalization constant and the main model parameters. Therefore, our model simulation scripts do not have to return anything, as all relevant parameters are defined already in the analysis scripts in the `webppl/phywpp1/examples/` folder. However, readers may well be interested in sampling the outcome of a diversification process along the tree. For instance, it may be desirable to analyze parameters such as the number and location of change events on different lineages, or the mean speciation rate for individual branches in the tree. To facilitate such analyses, we give an example model script for the ClaDS2 model returning the entire reconstructed tree, with descriptions of the outcome of the simulation process for each branch and node in the tree in extended Newick format. This script is found in the `webppl/phywpp1/models/` folder.

5.5 Birch model scripts

Birch is an object oriented probabilistic programming language. It uses more concise syntax than WebPPL for the probabilistic constructs. For example, the `assume` statement in the form

```
x ~ Exponential(1);
```

is used to express that a random variable (x in the example above) is distributed according to a given probability distribution (an exponential distribution with rate 1). Execution of such a statement depends on whether the variable has a value or not. If it has, its behavior is equivalent with an observe statement; if not, the variable is associated with the given distribution. Birch uses delayed sampling, so a concrete value might not be sampled until needed. Birch also supports explicit sample and observe statements. To draw a value from an exponential distribution with rate λ , we would write

```
t <- Exponential( $\lambda$ );
```

To state that an outcome of a random variable distributed according to a Poisson distribution with rate λ is 0, we would write

```
0 ~> Poisson( $\lambda$ );
```

The factor statement in WebPPL corresponds to `yield FactorEvent(log_factor)`. To simplify the diversification model definitions, we have defined two helper functions for commonly used `yield` statements.

```
yield duple();
```

corresponds to

```
yield FactorEvent(log(2));
```

and is used to account for the rotational factor at hidden speciation events. Similarly,

```
yield impossible();
```

is the same as

```
yield FactorEvent(-inf);
```

and it is used when simulated side branches resulting from hidden speciation do not go extinct, that is, when they are incompatible with the observed tree. Note that `yield impossible()` statement also ceases the execution of the particle.

As we have mentioned above, Birch is an object-oriented language and the models take advantage of this. For instance, the CRBD model script defines a `CRBDModel` class, which is derived from a base class called `PhyModel`. Let us examine a somewhat simplified version of the `CRBDModel` class definition (Algorithm 4), to see how it compares to the WebPPL script.

Algorithm 4 CRBDModel class definition in Birch (somewhat simplified)

```

1 class CRBDModel < PhyModel<PhyNode, PhyParameter> {
2    $\lambda_k$ :Real;
3    $\lambda_\theta$ :Real;
4    $\epsilon_{min}$ :Real;
5    $\epsilon_{max}$ :Real;
6    $\rho$ :Real;
7
8   fiber initial() -> Event {
9     super.initial();
10     $\theta.\lambda \sim \text{Gamma}(\lambda_k, \lambda_\theta)$ ;
11     $\theta.\epsilon \sim \text{Uniform}(\epsilon_{min}, \epsilon_{max})$ ;
12  }
13
14  fiber step() -> Event {
15    count:Random<Integer>; // number of (hidden) speciation events
16    count ~ Poisson( $\theta.\lambda * (\text{node.t\_beg} - \text{node.t\_end})$ );
17    for i in 1..Integer(count) {
18      t:Random<Real>;
19      t ~ Uniform( $\text{node.t\_end}, \text{node.t\_beg}$ );
20      simulateUnobserved(t);
21      yield duple();
22    }
23
24    0 ~> Poisson( $\theta.\lambda * \theta.\epsilon * (\text{node.t\_beg} - \text{node.t\_end})$ );
25
26    if node.isSpeciation() {
27      0.0 ~> Exponential( $\theta.\lambda$ );
28    }
29  }
30
31  fiber simulateUnobserved(t_beg:Real) -> Event {
32     $\Delta_d$ :Random<Real>; // waiting time until an extinction event
33     $\Delta_d \sim \text{Exponential}(\theta.\lambda * \theta.\epsilon)$ ;
34    t_d:Real <- t_beg -  $\Delta_d$ ;
35    if t_d < 0 {
36      // Species survived to the present time
37      yield impossible();
38    }
39  }

```

```

40     count:Random<Integer>; // number of speciation events
41     count ~ Poisson( $\theta \cdot \lambda * (t\_beg - t\_d)$ );
42     for i in 1..Integer(count) {
43         t:Random<Real>;
44         t ~ Uniform(t_d, t_beg);
45         simulateUnobserved(t);
46     }
47 }
48 }

```

The `CRBDModel` class is derived from the class `PhyModel`, which is a templated class. The base class takes care of tasks that are common to all diversification models, such as walking over the tree. This is analogous to the recursive calls in the `simTree` function in the WebPPL script (Algorithm 3), which also walk over the branches in the tree. At the top of the class definition, the member variables and their types are declared. These are the parameters of the prior distributions for the model variables λ and ϵ . The parameters are assigned specific values when the class is instantiated in connection with running the program. The inference settings and input values for the analyses are in the `config/crbd.json` file and in each of the `input/<name of tree>.json` input files.

Instead of member functions, the class defines several member fibers. A fiber (also known as a coroutine) is similar to a function, but the execution might be paused (e.g., to resample the particles) and resumed. The `initial` fiber initializes the simulation by assuming λ and ϵ to be distributed according to the appropriate priors. Note that these model variables are packaged inside an object called θ .

The `step` fiber corresponds to the `simBranch` function in the WebPPL script. In Birch, we use a different method for simulating the speciation and extinction events than in WebPPL. Rather than drawing the waiting times between hidden speciation events, we use the fact that the number of hidden events is described by a Poisson distribution, and the event positions are uniformly distributed over the branch length. This simulation method is faster than drawing each of the waiting times. In the line

```
 $\theta \rightsquigarrow$  Poisson( $\theta \cdot \lambda * \theta \cdot \epsilon * node.branch\_length$ );
```

we condition on the fact that there are 0 extinction events on the branch (recall that $\mu = \lambda\epsilon$). In WebPPL, we used a `factor` statement with the appropriate probability instead, which is an alternative way of accomplishing the same thing. Finally, in the line

```
 $\theta \cdot \theta \rightsquigarrow$  Exponential( $\theta \cdot \lambda$ );
```

we condition on there being a speciation at the end of the branch (if it ends in an interior node). Equivalently, we could have factored in $\log \lambda$, as we did in WebPPL, with a `yield` statement.

The `simulateUnobserved` fiber corresponds to the `goesExtinct` function in WebPPL. However, here we first simulate the time until the branch goes extinct. If the branch does not go extinct, we set the weight to zero, effectively killing off the simulation. If it does go extinct, we simulate the hidden speciation events along the branch, and call `simulateUnobserved` recursively for each of those events.

The code described above is subject to change, as Birch is developing rapidly. However, this section illustrates the basic Birch features, and how they can be used to code diversification models efficiently. Hopefully, it also sheds additional light on general PPL concepts, as it gives alternative but equivalent ways of coding some model elements compared to the WebPPL scripts we have seen previously.

6 Inference

In this section, we provide additional details on the non-standard algorithms we used to allow efficient PPL inference on phylogenetic diversification models.

6.1 Alignment

The encoding of the CRBD model given in Section 5.2 is rather natural—it is simply a description of the birth-death process, with a few calls to `factor` to correct for some probability effects that we do not model explicitly. Unfortunately, the default SMC algorithm implemented in WebPPL is quite inefficient for this naive implementation of the birth-death process. The algorithm *always* resamples particles (simulations are called *particles* in the SMC algorithm) at calls to `factor` and `condition`. Since, for every execution of the program, there is a different number of hidden speciation events on each branch in the observed tree, this will cause the SMC particles to get out of sync at resampling points. Particles that have few hidden speciation events may reach the end of the simulation long before particles that have many hidden speciation events. Thus, if we always resample at hidden speciation events, we will be comparing particles that can be at very different points in the simulation.

Intuitively, one might expect that it would be better to compare the particles only when they reach the same points in the probabilistic program. We call this *alignment* of the SMC resampling points. In the diversification models, we could, for instance, make sure that the resampling occurs only at the branching points in the observed tree. To explore this idea, we “tricked” the SMC algorithm in WebPPL to align the resampling points by introducing a few modifications to the birth-death simulation in the `simBranch` and `simTree` functions, as illustrated in the code below (compare to the naive CRBD simulation presented above in Algorithm 3):

Algorithm 5 A complete WebPPL script for simulating CRBD.

```
1 var simBranch = function( startTime, stopTime, lambda, mu )
2 {
3   var t = exponential ( {a: lambda} );
4
5   var currentTime = startTime - t;
6
7   if ( currentTime <= stopTime )
8     return 0.0;
9
10  var sideDetection = crbdGoesUndetected( currentTime, lambda, mu )
11  if ( sideDetection == false )
12    return ( -Infinity )
13
14  return simBranch( currentTime, stopTime, lambda, mu )
15    + Math.log( 2.0 );
16 }
17
18 var simTree = function( tree, parent, lambda, mu )
19 {
20   var lnProb1 = - mu * ( parent.age - tree.age );
21
22   var lnProb2 = ( tree.type == 'node' ? Math.log( lambda ) : 0 );
23
24   var lnProb3 = simBranch( parent.age, tree.age, lambda, mu );
25
26   factor( lnProb1 + lnProb2 + lnProb3 )
27
28   if ( tree.type == 'node' )
29   {
30     simTree( tree.left, tree, lambda, mu )
31     simTree( tree.right, tree, lambda, mu )
32   }
33 }
```

Specifically, we need the WebPPL SMC implementation to skip the resampling induced at the calls to `factor` and `condition` within `simBranch` in the naive model script. We achieve this by replacing the `factor` and `condition` statements in the `simBranch` function by code that accumulates the weight and returns it to `simTree`. The accumulated weight is then passed as an argument to `factor` in `simTree`, after the entire branch has been processed, triggering resampling at this point. The `factor` statement is also passed the probability of no extinction on the branch (`lnProb1`), and the likelihood of a speciation at the end of the branch, if it is an interior branch in the observed tree (`lnProb2`). Note that, to improve efficiency, we immediately return `-Infinity` in `simBranch` if a call to `goesExtinct` returns false, since there is no need to continue the recursion if this occurs. By modifying the simulation script in this way, the SMC particles stay in sync. There are no triggers of resampling in the `simBranch` recursion, so resampling is always performed in `simTree`, in between processing branches of the observed tree.

Simulations on a few example trees of varying sizes confirm that this indeed improves SMC efficiency on diversification models considerably (Supplementary Figure 3). The larger the tree, the more important it is for SMC performance to align the resampling points in this way. Ideally, one should not have to manipulate model scripts in the way described above; alignment should be applied automatically when it improves SMC efficiency. This is an idea that we are exploring within the TreePPL project. The goal is to analyze the potential performance gains induced by resampling, and then apply it intelligently either in the compiler and/or the language runtime. We separately present a static analysis for automatic alignment of programs³⁹. Note that alignment is not *guaranteed* to improve accuracy—in certain cases, it might actually degrade performance. However, for all models considered here, alignment is beneficial.

6.2 Delayed sampling

Probabilistic computations involve not only simulation and observation, as represented by the `sample` and `observe` statements in a PPL, but also such computations as marginalization, enumeration, and conjugate updating.

Consider the following joint distribution between two variables x and λ :

$$p(x, \lambda) = p(x | \lambda)p(\lambda), \tag{11}$$

where the two factors on the right are encoded in the probabilistic program as, for example:

$$\begin{aligned} \lambda &\sim \text{Gamma}(1, 1), \\ x &\sim \text{Poisson}(\lambda). \end{aligned}$$

We may wish to compute the marginal distribution of x :

$$p(x) = \int p(x | \lambda)p(\lambda) d\lambda, \tag{12}$$

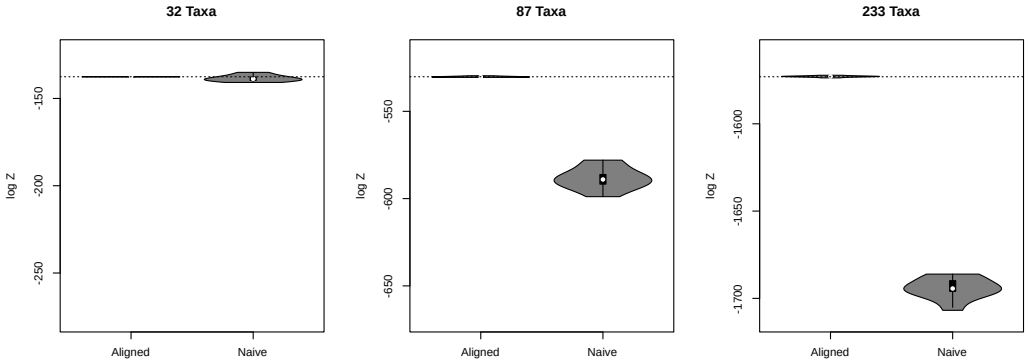


Fig. 3 A comparison of the precision in the estimated normalization constant between naive and aligned CRBD. Left: 32-taxon tree (Bisse_32). Center: 87-taxon tree (Cetaceans_87). Right: 233-taxon tree (Primates_233). SMC inference with 10,000 particles in WebPPL. Dotted line: exact analytical solution. Parameters: $\lambda = 0.2$, $\epsilon = 0.5$, complete sampling of leaves assumed ($\rho = 1$).

or, given a value of x , compute the posterior distribution over λ :

$$p(\lambda | x) = \frac{p(x | \lambda)p(\lambda)}{p(x)}. \quad (13)$$

Evaluations such as these can be performed analytically for random variables with a conjugate relationship (such as the gamma-Poisson relationship in the example above), or for discrete random variables where all possible outcomes can be enumerated. This can improve the performance of inference by, for example, reducing the variance in statistical estimators, such as that for the marginal likelihood.

Delayed sampling⁶ is a particular heuristic that may be employed by a PPL to identify and leverage such situations to improve inference outcomes. It does so in a manner that produces correct results, even for programs with stochastic branches and unbounded recursion as may be encountered in Turing-complete programming languages. It is not necessary for the programmer to painstakingly code such computations by hand.

We have used delayed sampling extensively in this work, substantially reducing the variance in marginal likelihood estimates. In particular, for Poisson processes on trees, gamma prior distributions over rates are conjugate either to the Poisson-distributed number of events in a given time interval, or the exponentially-distributed time between events. These rate parameters are then automatically marginalized out by delayed sampling, substantially reducing variance in the marginal likelihood estimate for these models. The same approach to handling parameters is used in Kudlicka et al.⁴⁰ and Wigren et al.⁴¹. Delayed sampling is only available in Birch at this point.

6.3 Alive particle filter

The resampling step in SMC amounts to drawing N samples (with replacement) from the current set of N particles with probabilities proportional to their weights. While simulating the evolution of unobserved side branches, if any species survives to the present day (and is sampled), the weight of the particle must be set to 0. This leads to sample impoverishment—there are fewer particles to choose from during resampling. In extreme cases, where all particles have zero weight, there are no particles to choose from at all, and the algorithm fails. This can be a serious problem for SMC inference on diversification models when the likelihood of extinction of side branches is low. For instance, this can occur if the net diversification rate ($\lambda - \mu$) is high.

The extended alive particle filter⁴⁰—the development of which from the initial version of this algorithm⁴² was inspired by phylogenetic diversification models—solves these two problems by replacing the particles with zero weights with new samples drawn from the particle set at the previous time step (again with probabilities proportional to the particle weights at that time) and repeating the propagation step (the simulation from the previous resampling point until the current resampling point). This replacing procedure is repeated until the weights of all particles are positive. Note that in order to estimate the marginal likelihood without bias, one needs to repeat this procedure for one additional particle. However, with a reasonable number of particles, this extra computational cost is negligible, and we therefore applied the alive particle filter to all analyses. The alive particle filter is only available in Birch at this point.

6.4 Tree orientation

During the course of the study, we discovered that the orientation of the nodes in the observed tree can have a noticeable influence on the efficiency of SMC inference for some trees. The effect appears to be associated with highly imbalanced trees, which may

be oriented such that left and right subtrees systematically have different properties. A depth-first SMC algorithm can apparently become misled by the imbalance between left and right descendants in such trees, so that early resampling events can select particles that do not do well towards the end of the simulation, decreasing the quality of the final estimate. We found that orienting all nodes such that the descendant branch with the shortest subtree length was always processed first solved this problem. Thus, all trees were reoriented in this way before final analyses in Birch.

7 Verification

We performed a wide range of experiments to verify that the model scripts are correct. For all tests involving WebPPL or third-party software, the full set of experiments—including the source code, data, graphs and reports—can be found in the directory verification of the `phywppl` package. The verification experiments involving Birch were performed by changing the input files and/or models to fix the values of selected parameters. The results from these experiments are included in the above-mentioned directory, together with the results from the experiments involving WebPPL.

Here, we only present a summary of the experiments. They all use the 32-taxon example tree, which we provide as one of the builtin trees in the `phyjs` package. The tree has been previously used as an example in diversification model papers; it is originally from the Mesquite software¹⁹ but does not appear to have been published separately. Only scripts adapted for aligned SMC inference were used in the verification experiments.

The experiments are based on several lines of attack. In the first round of tests, we used the fact that there are analytical solutions for the likelihood of the simple diversification models (CRB, CRBD, TDB and TDBD) under specific parameter values. Thus, we could verify that the normalization constant computed by SMC from our explicit simulation scripts for the same models (the scripts that simulate the process along the tree instead of calling the likelihood function) matched the corresponding analytical likelihoods for a wide range of specific parameter values. These tests are important because the explicit simulation scripts for the simple models served as templates for the scripts describing the more complex models.

The second round of tests were based on the observation that the more complex diversification models (ClADS0, ClADS1, ClADS2, LSBDS and BAMB) all collapse to simpler models with analytically known likelihoods under specific parameter settings. This allows us to verify that the normalization constant computed from the scripts for the complex models matched the corresponding analytical likelihoods for select points in parameter space.

For other points in parameter space, we cannot verify the scripts for the more complex models against analytical likelihoods, but we can use other approaches to test their correctness. For instance, the WebPPL and Birch scripts for the complex models were implemented independently by different co-authors of this paper, and the inference algorithms in WebPPL and Birch were also different and based on independent implementations. In the third round of tests, we verified that the WebPPL and Birch scripts for the complex models gave the same normalization constant for a grid of parameter values despite these differences.

The fourth round of tests took advantage of the independent implementations available in third-party software for the ClADS models⁴³ and for LSBDS²⁹. We verified that our scripts for these models resulted in the same estimates of the likelihood as these implementations for a select set of parameter values, despite being based on entirely different code bases and computational strategies.

Third-party software also exists for BAMB³⁸ but it does not compute correct likelihoods for the model³², so it cannot be used to verify our scripts. However, the BAMB model collapses to the LSBDS model when all $z^i = 0$. We therefore verified that our BAMB model script results in the same likelihood estimates as the LSBDS model script under select parameter values matching this constraint but lacking analytical solution.

7.1 Simple models against analytical likelihoods

All simulation scripts for simple models (CRB, CRBD, TDB and TDBD) generated normalization constant estimates that matched the corresponding analytical likelihoods very closely. We observed some variance in the estimates for high λ values, but these parameter values have low likelihood and are thus less important for inference (Supplementary Figure 4). All $\lambda \times \epsilon$ combinations for two values of ρ : $\rho = 0.5$ (incomplete sampling in the order of magnitude of the ρ -range for the bird trees) and $\rho = 1$ (complete sampling) have been checked.

7.2 Complex models against analytical likelihoods

All model scripts for advanced diversification models (ClADS0, ClADS1, ClADS2, LSBDS and BAMB) generated normalization constant estimates that matched analytical likelihoods under parameter settings for which closed solutions exist (Supplementary Figure 5, 6). Again, we checked all $\lambda \times \epsilon$ combinations for two values of ρ : $\rho = 0.5$ (incomplete sampling in the order of magnitude of the ρ -range for the bird trees) and $\rho = 1$ (complete sampling).

7.3 Birch and WebPPL cross-verification

Under parameter and prior settings for which closed solutions do not exist, the independently developed Birch and WebPPL scripts for advanced diversification models resulted in matching normalization constant estimates. Birch estimates were slightly more precise than WebPPL estimates (Supplementary Figure 7) but this is expected given the more powerful inference algorithms used by Birch. In the tests illustrated in the figure, $\rho = 1$ is assumed. To verify that our code is correct also for $\rho < 1$, we ran

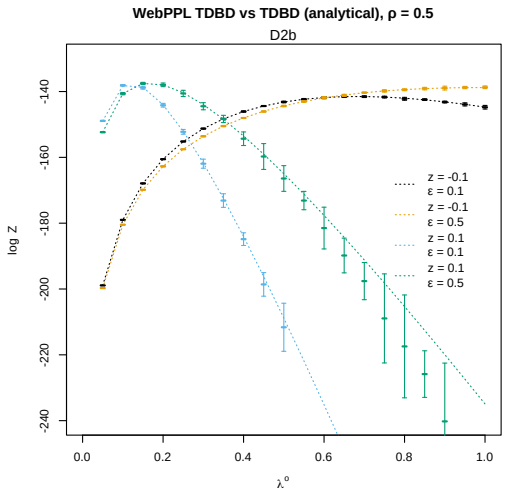
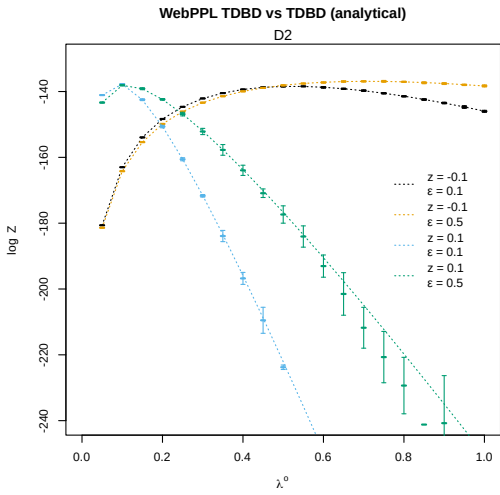
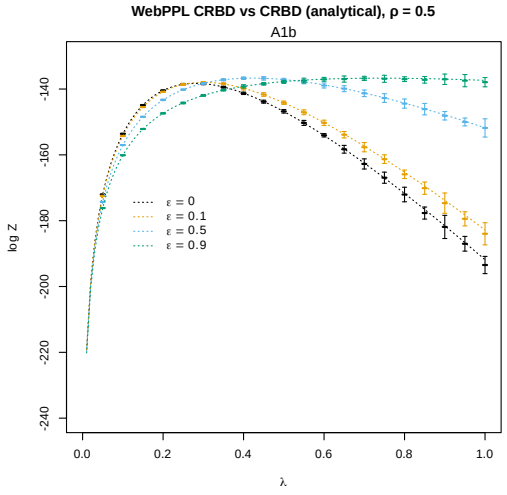
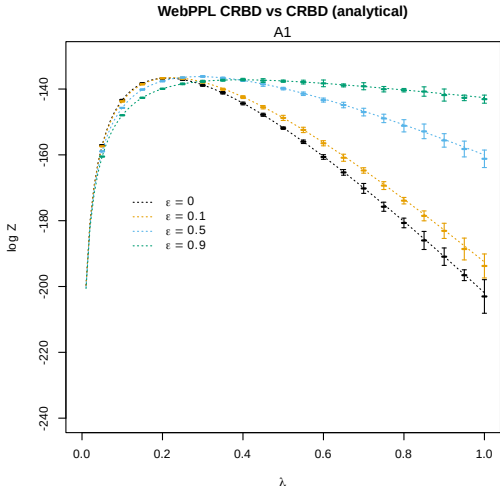


Fig. 4 Verification of explicit simulation scripts (WebPPL) for simple diversification models: normalization constants match analytical likelihoods for select parameter values. Error bounds: ± 2 standard deviations. Experiment codes in the GitHub repository indicated under the main title.

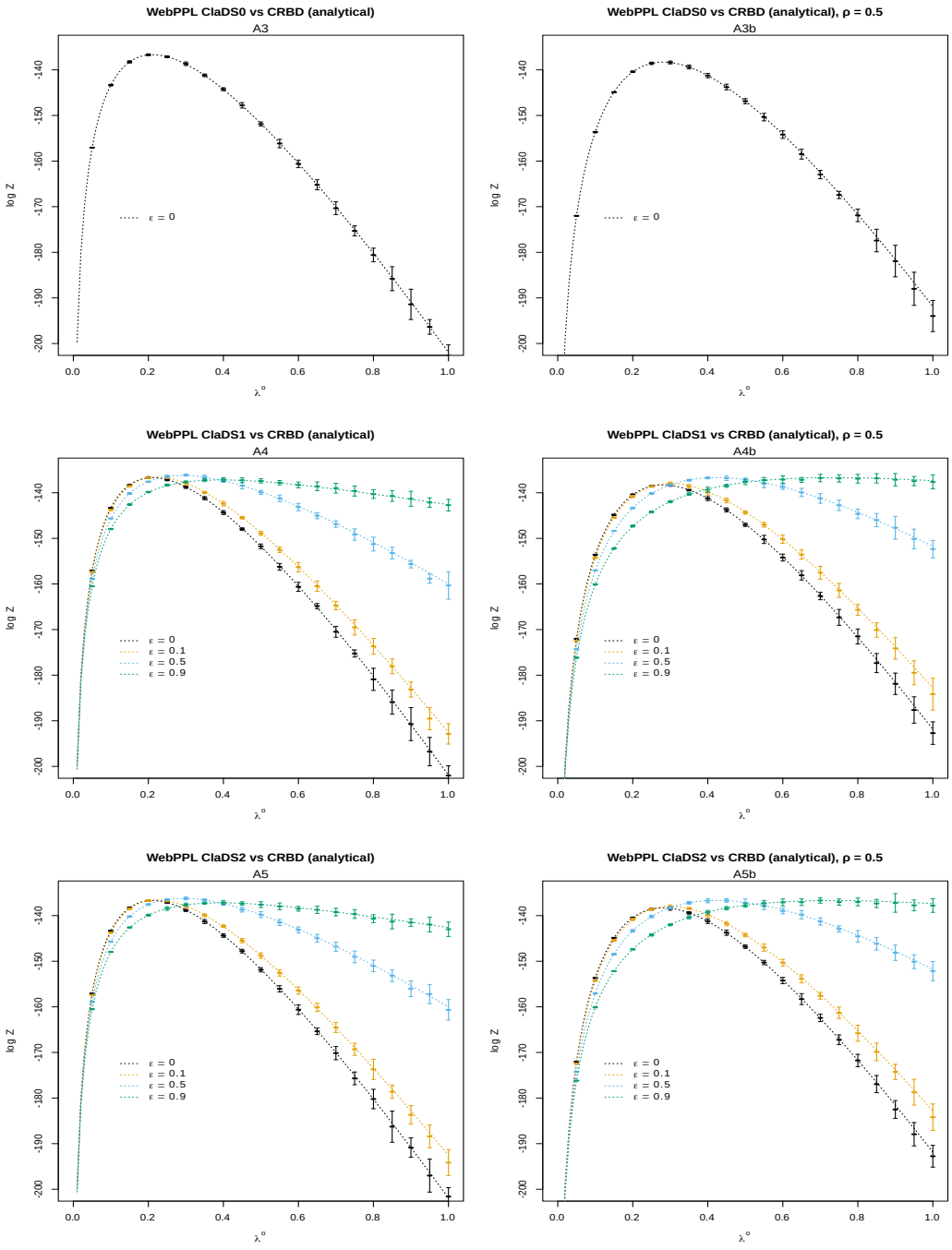


Fig. 5 Verification of WebPPL simulation scripts for the ClADS[0-2] models: normalization constants match analytical likelihoods for select parameter values. Error bounds: ± 2 standard deviations. Experiment codes in the GitHub repository indicated under the main title.

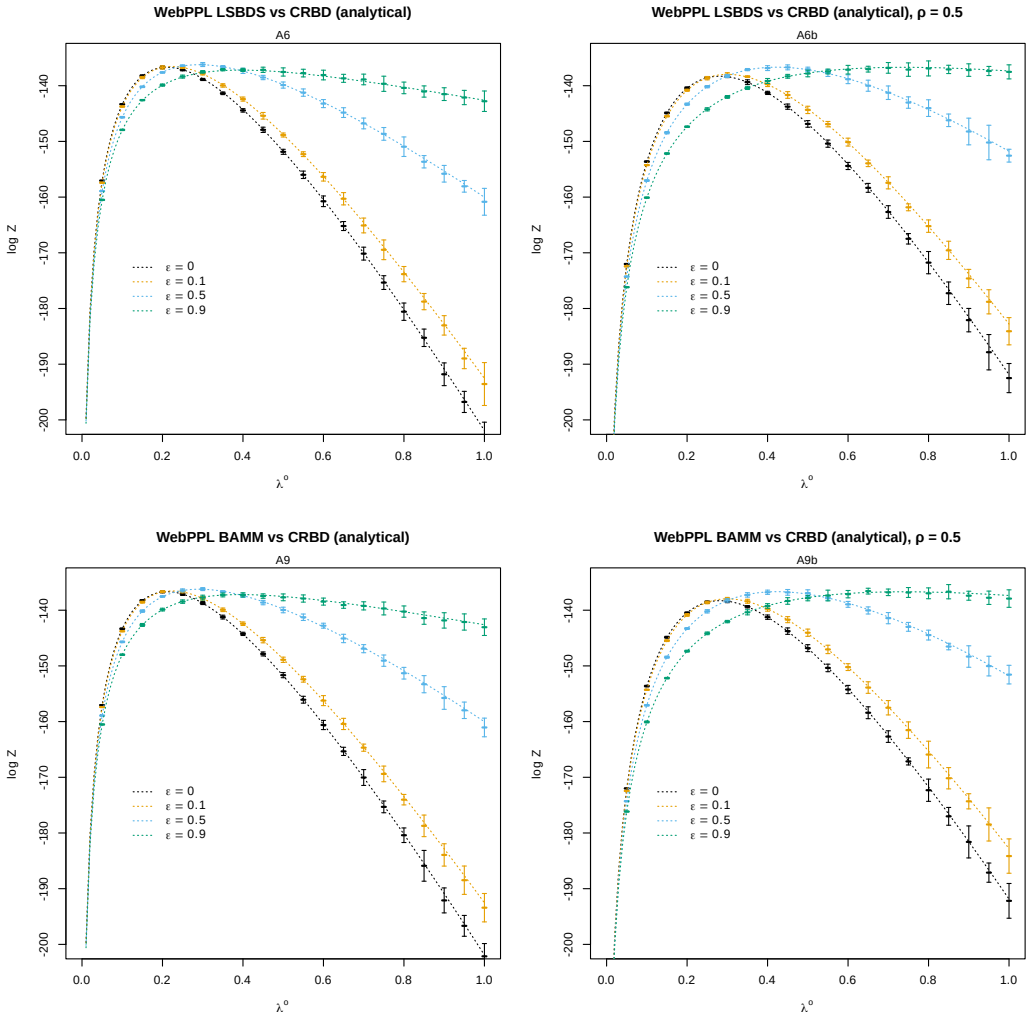


Fig. 6 Verification of WebPPL simulation scripts for lineage-specific diversification models (complex models): normalization constants match analytical likelihoods for select parameter values. Error bounds: ± 2 standard deviations. Experiment codes in the GitHub repository indicated under the main title.

Table 5 Cross-verification of WebPPL and Birch for $\rho < 1$.

Model	ρ	WebPPL		Birch	
		mean log Z	std. dev. log Z	mean log Z	std. dev. log Z
BAMM	0.1	-146.839	0.623	-146.608	0.279
BAMM	0.5	-139.612	0.198	-139.544	0.089
ClaDS0	0.1	-152.348	0.921	-150.540	0.663
ClaDS0	0.5	-142.506	0.450	-142.528	0.173
ClaDS1	0.1	-153.957	2.003	-151.245	0.635
ClaDS1	0.5	-143.422	0.351	-143.385	0.164
ClaDS2	0.1	-152.426	1.578	-151.747	0.915
ClaDS2	0.5	-143.085	0.315	-143.000	0.161
CRBD	0.1	-172.826	0.093	-172.795	0.089
CRBD	0.5	-143.259	0.083	-143.314	0.077
LSBDS	0.1	-172.794	0.120	-172.819	0.095
LSBDS	0.5	-143.361	0.075	-143.321	0.032
TDBD	0.1	-145.567	0.989	-145.042	0.063
TDBD	0.5	-139.039	0.263	-139.028	0.000

several additional grid point experiments as summarized in Table 5. All experiments in the table were conducted on the 32-taxon tree using the standard priors, except for $\lambda = 0.2$, $\epsilon = 0.5$, and ρ , for which point values were used instead. For TDBD, the Birch implementation uses the analytical solution.

7.4 Verification of ClaDS models against RPANDA

Verification of the probabilistic programs and PPL inference algorithms described in this paper against the reference RPANDA implementation of the ClaDS models is quite involved, and would not have been possible without extensive help from the author of the ClaDS code in RPANDA (Odile Maliet), as computation of the likelihoods with RPANDA is not part of its public API. RPANDA computes the likelihood for points in parameter space where all the initial λ^i values for the branches in the reconstructed tree are known, as well as the λ^o value, pertaining to the MRCA of the tree. The likelihood in RPANDA is also conditioned on specific values of the model parameters α and σ , as well as on μ (for ClaDS1) or ϵ (for ClaDS2). The ClaDS0 likelihood function in RPANDA is based on analytical equations, while the ClaDS1 and ClaDS2 functions are based on numerical approximations using a variety of techniques. In addition, the functions only give the density up to a proportionality constant, further complicating direct comparisons with our scripts.

The RPANDA setup means that the PPL scripts have to condition on specific values for all of the model parameters, including the initial λ values for all branches, to emulate the RPANDA likelihood computations. To be able to conduct the verification experiments, we decided to use a fixed value λ_f for λ^o and all λ^i parameters of the model in our WebPPL scripts; we did not attempt to perform these verification experiments in Birch. We then chose a range of λ_f values, and explored these points in parameter space under some specific values of α , σ and μ (for ClaDS1) and ϵ (for ClaDS2). Likelihoods for the same points in parameter space were then computed in RPANDA with the analytical likelihood function (for ClaDS0) and the numerical solvers (for ClaDS1 and ClaDS2). In the git repository accompanying the paper, we provide both the WebPPL scripts emulating the RPANDA computations and the R scripts we used to compute likelihoods for the corresponding points in parameter space with RPANDA.

For ClaDS0, the initial experiments showed that the likelihood function in RPANDA computes densities that very closely match the densities expected for oriented and unlabeled trees. Thus, we concluded that the proportionality constant for the ClaDS0 likelihood function in RPANDA is the same as the conversion factor from densities on oriented and unlabeled trees to densities on labeled, unoriented trees. This factor is $L_p = \log(2^{(n-1)}/n!)$, where n is the number of leaves in the tree (see Section 3.2).

When controlling for this, the likelihoods estimated by WebPPL for ClaDS0 are consistent with those computed by RPANDA (Supplementary Figure 8). For points in parameter space where ClaDS1 and ClaDS2 collapse to ClaDS0, that is, for points where $\mu = \epsilon = 0$, likelihoods estimated by WebPPL and RPANDA are also very similar. The same is true for small values of λ and μ in ClaDS1, and for small values of λ and ϵ in ClaDS2. For larger values, RPANDA apparently overestimates the likelihood for both models, and there are also some apparent discretization effects at very high values of λ . We tried to examine the effects of these inaccuracies in RPANDA on the posterior estimates of the ClaDS1 and ClaDS2 model parameters for the test tree, but were unable to get sufficiently good MCMC convergence in RPANDA to allow meaningful analysis of these results.

7.5 Verification of LSBDS against RevBayes

In the current implementation of LSBDS in RevBayes (the SCM algorithm), the likelihoods are computed by discretizing the λ and μ priors. Transitions happen by “jumping” from one pair of discrete values of λ and μ to a different pair. We discovered that λ and μ are coupled when these jumps are made: i.e., the discrete vectors representing the prior distributions f_λ and f_μ have to be of the same length and, when a jump happens, a single new array index is chosen for both the λ and the μ vector. Thus, usually,

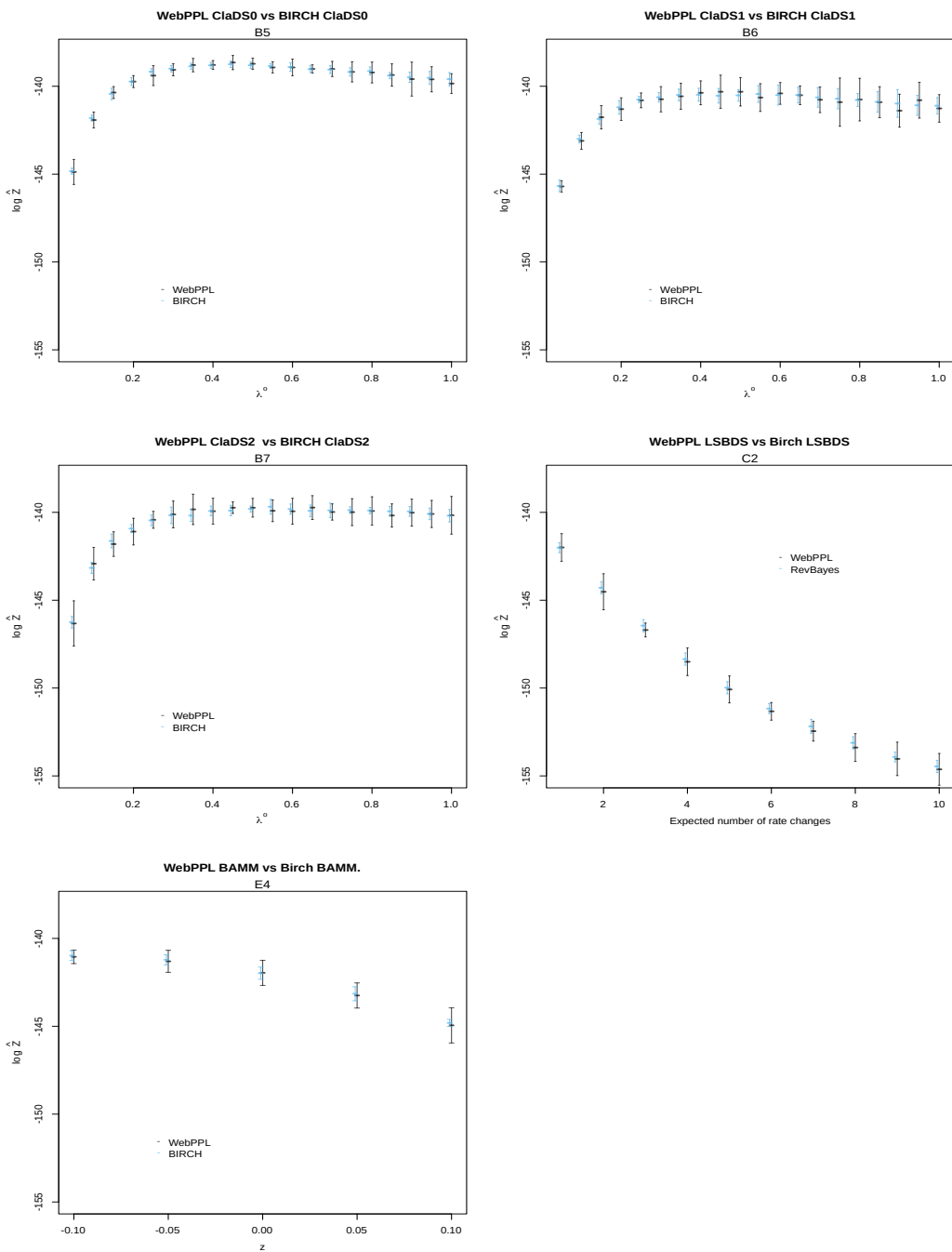


Fig. 7 Verification that the WebPPL and Birch scripts for lineage-specific diversification models generate normalization constants that match each other for select parameter values.

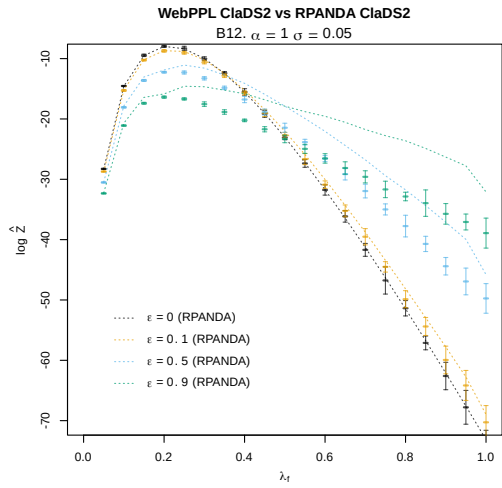
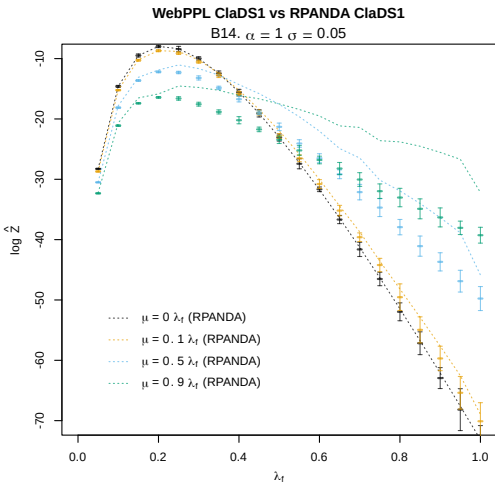
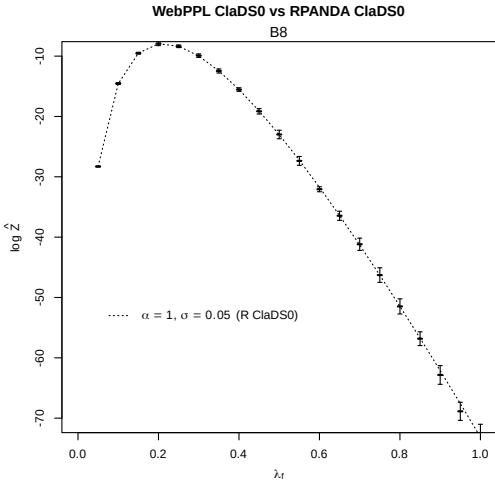


Fig. 8 Verification of the likelihoods computed by WebPPL in programs emulating RPANDA against likelihoods computed by RPANDA for the ClaDS models.

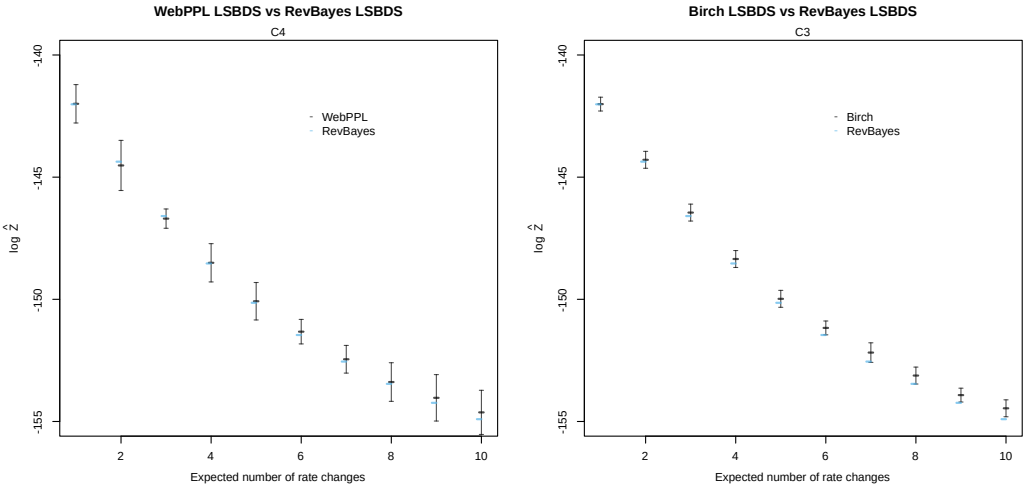


Fig. 9 Verification that the WebPPL and Birch scripts for the LSBDS model generate normalization constant estimates that match the numerically estimated likelihood computed by RevBayes.

the RevBayes LSBDS examples^h fix λ but discretize μ (or vice-versa). However, it is possible to discretize both λ and μ and then expand the two arrays so that all possible combinations of λ and μ values appear when sweeping both vectors simultaneously with a single array index. This has to be done manually.

We verified our LSBDS scripts against RevBayes for specific values of η and integrated out $\lambda \sim \text{Exponential}(1)$ and $\mu = \epsilon\lambda$, where $\epsilon \sim \text{Uniform}(0, 1)$ using $k = 10$ rate categories for both λ and μ . We implemented the appropriate vector of parameter values manually, as described above. The RevBayes scripts used in the verification experiments are provided in the git repository accompanying the paper.

Under these settings, both the WebPPL and Birch scripts for the LSBDS model generate normalization constant estimates that match the likelihoods computed by RevBayes (Supplementary Figure 9). As observed previously in several experiments, Birch provides slightly more precise estimates of the normalization constant than WebPPL.

7.6 Verification of BAMB against LSBDS

There is no third-party software implementing BAMB that we can verify the WebPPL and Birch scripts against. However, we can use the fact that BAMB collapses to LSBDS when all z^i values approach 0. Under these conditions, and when integrating out the other model parameters, both the WebPPL and Birch simulations scripts for BAMB produce the same normalization constants as the corresponding LSBDS scripts (Supplementary Figure 10),

8 Empirical data

For the empirical analyses illustrating PPL inference for phylogenetic diversification models, we used the bird trees analyzed previously for the ClaDS2 model by Maliet et al.³⁰. The trees originate from an earlier study inferring a global timed phylogeny of birds⁴⁴. Specifically, clades with 50 or more leaves (excluding outgroups) from the earlier study were selected in the ClaDS2 study³⁰ and post-processed to remove outgroups and to rescale branch lengths to time units (myr). Also, only species for which there is molecular data have been included in the trees analyzed by Maliet et al.³⁰; consequently the authors calculated a sampling fraction (ρ) by dividing the number of tips in the trees computed for species with genetic data by number of tips in the corresponding complete tree (private correspondence).

We downloaded these post-processed trees from the repositoryⁱ accompanying the ClaDS2 paper and extracted the corresponding sampling fractions ρ .

The trees were converted from binary R data (RData) to text format (Nexus) with the ape package. The Nexus files were then converted to PhyJSON with the nexus2phyjson tool that we provide¹⁸. Next, the PhyJSON trees were reoriented to avoid any systematic left-right imbalances in the original trees that could have a negative effect on inference (see Section 6.4). The resulting PhyJSON trees were then used as input data for the WebPPL and Birch analyses.

^h<https://github.com/hoehna/birth-death-shift-analyses>

ⁱhttps://github.com/OdileMaliet/ClaDS/tree/master/birds_MCC_results

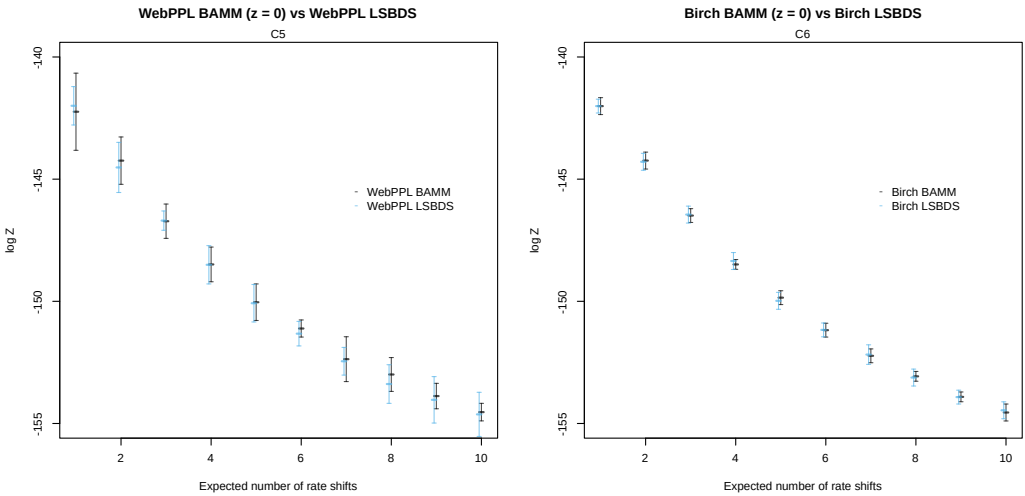


Fig. 10 Verification that the WebPPL and Birch scripts for the BAMM model generate normalization constant estimates that match those of the corresponding scripts for the LSBDS model for some points in parameter space where the BAMM model collapses to LSBDS.

There are 42 bird clades in the Jetz et al.⁴⁴ study with more than 50 species excluding outgroups. However, we discovered that two of the trees, P2 and Scolopaci, have negative branch lengths. Rather than introducing arbitrary corrections for the negative branch lengths, we excluded these trees from further analysis. The remaining 40 bird trees are summarized in Table 6. The original names of the bird clades⁴⁴ are rather cryptic. Here, we named the clades after the family (or other higher taxon) to which most members belong according to the taxonomic classification used in the original bird study⁴⁴. If a family is split between two clades, the clades are numbered 1 and 2. A '-' sign after the family name indicates that some members of the family are not included in the clade; a '+' sign indicates that the clade includes some members of other families. Four of the trees in the repository accompanying the ClaDS paper³⁰ are mislabeled there: Caprimulgidae is incorrectly labeled CC7, CC4 is labeled Cathartidae, CC7 is labeled CC5CC6B, and CC8 is labeled CC5CC6C.

The size of the trees vary from 54 (Alcedinidae) to 316 leaves (Tyrannidae+), and the ages from 12.5 Ma (Thraupidae1+) to 66.6 Ma (Cuculidae). The fraction of species included in the trees, that is, the sampling fraction ρ , varies from 0.43 (Columbidae) to 0.91 (Hirundinidae). The tree shapes are depicted in Supplementary Figures 11, 12.

9 Efficiency of inference algorithms

In this section, we provide detailed information about the efficiency of the inference algorithms, taking into account both the quality of the samples of the posterior distribution, and the computational resources needed in obtaining those samples. Specifically, we take advantage of the most obvious approach to measuring the quality of a Monte Carlo inference procedure: we repeat the analysis many times (by running the corresponding program multiple times), and then assess the consistency of the estimates.

We focus on the normalization constant estimates across independent Monte Carlo analyses, as the normalization constant is influenced by all components in the model, including priors, latent variables and data. Thus, the consistency of the normalization constant estimates should provide a good overall estimate of the efficiency of the inference procedure.

Clearly, the more computational resources we invest in obtaining an estimate of the normalization constant, the better that estimate will be. When is the estimate good enough? This depends on how the estimate is to be used. Consider, for example, if the estimate is to be used within a pseudomarginal Metropolis-Hastings sampler. In this case, there is a trade-off between the quality of the normalization constant estimate and the overall efficiency of the MCMC procedure. The efficiency of the MCMC procedure (per time unit) will increase with the precision of the normalization constant estimate, but will decrease with the time required to obtain it. Given some reasonable assumptions, it turns out that the optimal choice is to target a standard deviation of 1.0 if the Metropolis-Hastings algorithm using the exact likelihood is efficient, and around 1.7 when it is not⁴⁵. This suggests that a normalization constant estimate with a standard deviation close to 1.0 is more than satisfactory for this kind of demanding application.

With this in mind, we use two additional diagnostics to assess the efficiency of our inference procedure: the relative effective sample size (RESS) and the conditional acceptance rate (CAR). The former assesses the effect of using the normalizing constant estimates as the weights for an importance sampler, the latter of using them as unbiased estimates of the marginal likelihood for a pseudomarginal sampler.

Table 6 Overview of the bird trees used for diversification analyses.

Tree	Clade (Jetz et al.)	Leaves	ρ	Age (Ma)	Notes
Accipitridae	Accipitridae	175	0.71	59.6	Hawks, eagles, kites and allies
Alcedinidae	Alcedinidae	54	0.57	34.9	Kingfishers
Anatinae	Anatinae	108	0.87	20.3	Dabbling ducks
Caprimulgidae	Caprimulgidae	57	0.61	57.3	Nightjars
Campephagidae-	CC4	70	0.85	30.1	Cuckooshrikes and allies
Charadrii	Charadrii	63	0.62	59.6	Waders
Columbidae	Columbidae	133	0.43	35.9	Pigeons and doves
Corvidae+	CC8	234	0.66	30.0	Crows, magpies, monarchs and allies
Cuculidae	Cuculidae	126	0.88	66.6	Cuckoos
Emberizidae-	P20b	125	0.77	14.8	Buntings
Estrildidae	P7	101	0.62	19.5	Estrildid finches
Fringillidae+	P10	123	0.63	25.4	True finches
Furnaridae	Furnaridae	205	0.67	19.9	Ovenbirds
Hirundinidae	S6	77	0.91	23.1	Swallows, martins and allies
Icteridae	P21	92	0.88	14.0	New World blackbirds, orioles and allies
Lari	Lari	127	0.84	24.6	Gulls
Malaconotidae+	CC7	80	0.55	31.4	Bushshrikes
Meliphagidae+	BC7	90	0.49	37.1	Honeyeaters
Muscicapidae+	M6	231	0.77	20.2	Old World flycatchers
Paridae+	S2	55	0.72	40.9	Tits
Parulidae+	P20a	111	0.89	17.2	New World warblers
Phasianidae	Phasianidae	131	0.73	27.2	Pheasants, partridges and allies
Picidae	Picidae	137	0.61	27.1	Woodpeckers
Procellariidae	Procellariidae	105	0.81	59.6	Shearwaters, fulmarine petrels and allies
Psittacidae1	Psittacidae1	111	0.65	33.2	True parrots (part)
Psittacidae2	Psittacidae2	118	0.72	34.9	True parrots (part)
Pycnonotidae+	S9	95	0.73	29.4	Bulbuls
Ramphastidae	Ramphastidae	81	0.65	32.2	Toucans
Strigidae	Strigidae	101	0.52	45.7	True owls
Sturnidae+	M4	130	0.87	24.9	Starlings, mockingbirds and allies
Sylviidae1+	S11	79	0.65	28.1	Warblers, parrotbills and allies (part)
Sylviidae2+	S7S8	93	0.79	24.3	Warblers, parrotbills and allies (part)
Thamnophilidae	Thamnophilidae	165	0.74	22.4	Antbirds
Thraupidae1+	P13P14P16	158	0.71	12.5	Tanagers (part)
Thraupidae2+	P17P18	139	0.89	13.7	Tanagers (part)
Timaliidae+	S13	180	0.49	21.0	Old World babblers
Trochilidae	Trochilidae	233	0.69	28.1	Hummingbirds
Troglodytidae+	M1	91	0.69	32.7	Wrens
Turdidae+	M5	134	0.86	21.7	Thrushes
Tyrannidae+	Tityranrest	316	0.69	33.6	Tyrant flycatchers

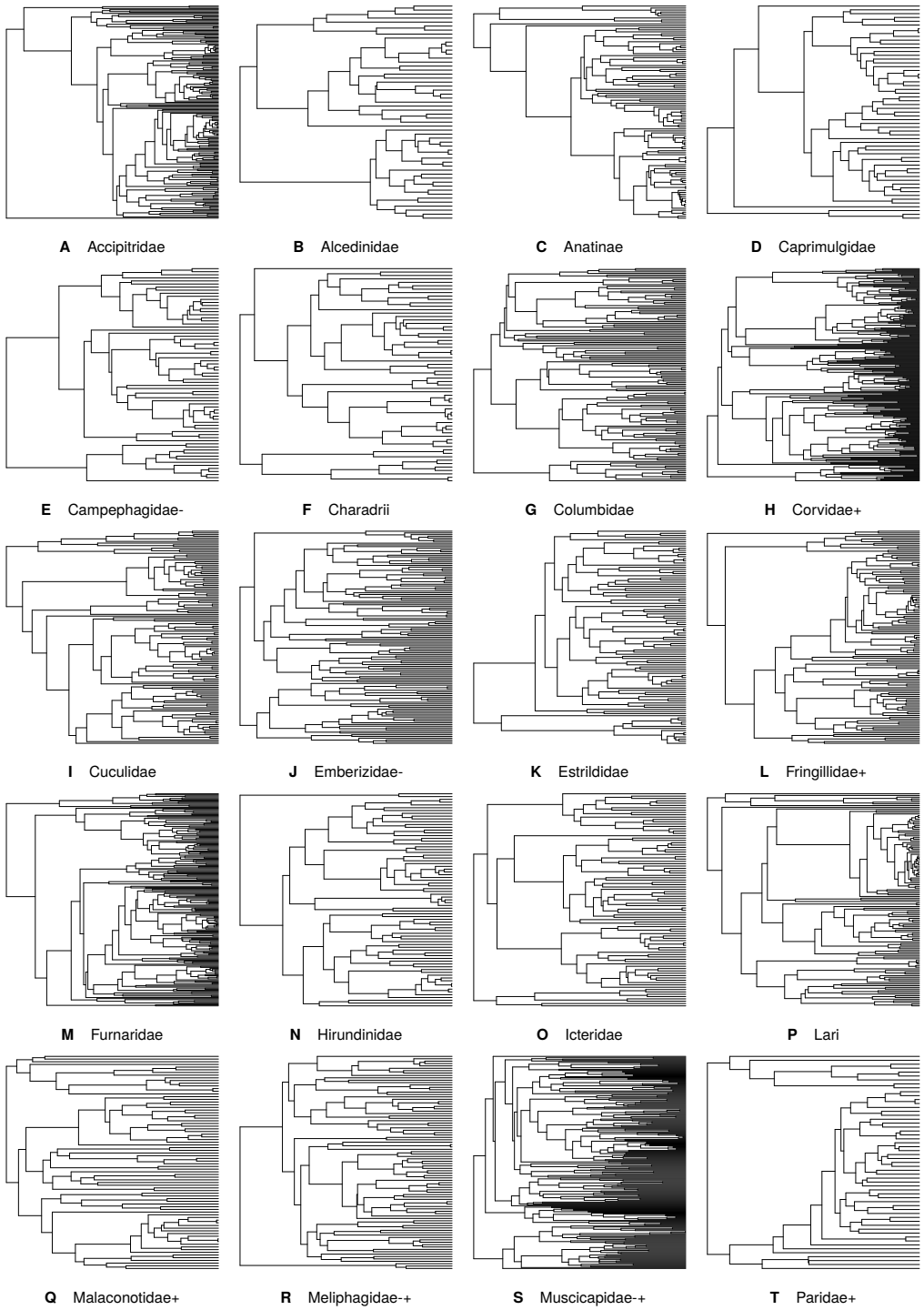


Fig. 11 Shape of the bird trees, part 1

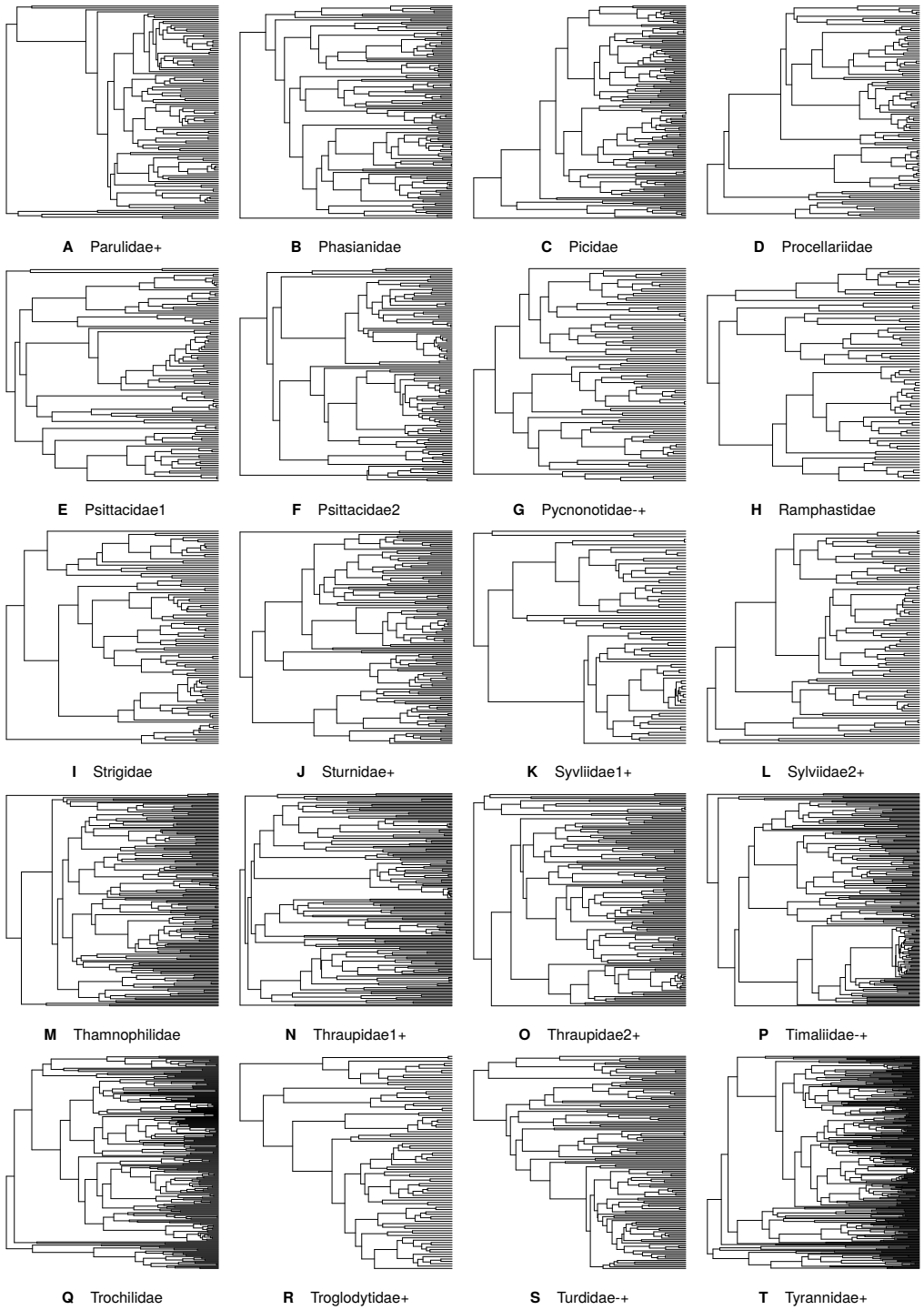


Fig. 12 Shape of the bird trees, part 2

Specifically, if we have N particles with normalized weights $\{w_1, w_2, \dots, w_N\}$, the effective sample size N_{eff} is computed as

$$N_{\text{eff}} = \frac{(\sum_i w_i)^2}{\sum_i w_i^2}. \quad (14)$$

This is known as Kish's ESS⁴⁶. We then compute the relative ESS (RESS) simply by dividing the ESS with the number of independent Monte Carlo estimates we had of the normalization constant. The RESS, which is a value on the interval $(0, 1]$, measures the efficiency of the Monte Carlo estimation procedure⁴⁰.

Like RESS, CAR is a value on the interval $(0, 1]$ ⁴⁷. It measures the expected drop in acceptance rate of a Metropolis-Hastings proposal due to errors in the estimate of the normalization constant, that is, similar to the logic we described above in establishing a quality criterion for the standard deviation. This theoretical acceptance rate is measured for each sampled point, as though the proposal distribution were shrunk to a Dirac δ distribution. The CAR is related to the standard deviation of the normalization constant estimate, but, like RESS, is a more direct measure of the impacts of that estimate on an inference procedure.

In the initial analyses, we used 10,000 particles in the importance sampling procedure (for the CRB(D) and TDB(D) models) and 5,000 particles in the SMC procedure with the alive particle filter (for the remaining models). The standard deviation of the normalization constant was well below 1.0 in all sequential importance sampling runs (Table 7). In the SMC runs, the standard deviation was usually close to or below 1.0 for all models except BAMM. However, we noted standard deviations above 2.0 in 2 of 40 trees for ClaDS1 (Muscicapidae+ and Thamnophilidae) and in 5 of 40 trees for LSBDS (Accipitridae, Muscicapidae+, Timaliidae+, Tyrannidae+ and Trochilidae). For BAMM, we increased the number of particles to 20,000, which brought the standard deviations down to more acceptable levels, even if we still noted values above 2.0 for 17 of 40 trees.

The RESS and CAR values largely reflect the standard deviation of the normalization constant estimates. However, we observed some SMC cases where the RESS and CAR values suggest that the sample is reasonably good, even though the standard deviation is high. Notable examples include the BAMM results for Corvidae+, Columbidae and Cuculidae, and the LSBDS results for Muscicapidae+ and Trochilidae.

All analyses of empirical data were run on Tetralith, the largest high-performance computing cluster at the National Supercomputer Centre (NSC), Sweden, a part of the Swedish National Infrastructure for Computing (SNIC). The cluster comprises 1908 nodes, each with two Intel Xeon Gold 6130 CPUs (each with 16 cores). There are 1832 nodes with 96 GiB and 60 nodes with 384 GiB of RAM.^j The operating system on the nodes is Linux (version 3.10.0) and the cluster uses SLURM (18.08.8) to schedule jobs. For each tree and model, we submitted an array of 10 single-core jobs with the memory limit set to 8 GiB (to avoid running out of memory for the largest trees), each running the respective Birch program 50 times. We used the latest development version of Birch^k and its standard library (as of June 12, 2020).

The median time (among 500 replicates) required to complete an importance sampling analysis (10,000 particles) using this hardware and analysis setup ranged from a few seconds to around one minute (Table 8). The SMC analyses (5,000 or 20,000 particles) were more demanding, requiring from around a minute to more than 50 minutes in extreme cases. The longest run times were usually associated with the BAMM model, for which we used four times as many particles (20,000) as for the other models. For models other than BAMM, the median run times rarely exceeded ten minutes. As one might expect, the largest trees were generally associated with the longest run times.

10 Extended results

The main purpose of the empirical analyses is to demonstrate the power of probabilistic programming in addressing inference problems in phylogenetics, not to advance the field of diversification studies. Nevertheless, there are several interesting patterns in the results that deserve attention and that may inspire further study. In this section, we present model likelihoods and posterior estimates of model parameters for all bird clades and diversification models (Supplementary Figures 13–22). The plots of posterior distributions should be interpreted in relation to the prior distributions for the corresponding regions of parameter space (Supplementary Figures 23). We structure the discussion of the results around several cross-cutting themes.

10.1 Conservative nature of Bayesian model tests

One of the most striking patterns across the bird trees, especially given the recent debate about the importance of accommodating lineage-specific diversification rates, is that simple birth-death models do so well in a Bayesian model comparison. In at least 16 of the 40 bird trees, there is no strong evidence against the simple CRB and CRBD models. In fact, in most of these cases, the simple models (either CRB(D) or TDB(D)) have the best normalizing constants. There are also some cases where the TDB(D) models do clearly better than the other models, significantly so in a couple of cases (Emberizidae- and Muscicapidae+, Supplementary Figures 15 and 17, respectively).

There is a clear correlation between the size of the tree and the outcome of the model comparison. Of the trees with less than 100 leaves, the CRB(D) models adequately describe the diversification process in a majority of cases, as indicated by Bayes factors. The largest trees lacking strong evidence of lineage-specific or slowing diversification have around 130 leaves (Columbidae, Cuculidae, Phasianidae and Sturnidae+; Supplementary Figures 14, 15, 18 and 20, respectively). Above that size,

^j<https://www.nsc.liu.se/systems/tetralith/>

^k<https://github.com/lawmurray/Birch>

Table 7 Diagnostics for the normalization constant estimates obtained across 500 runs for each tree and model. The first line in each cell shows the mean and standard deviation of the normalization constant estimates. We also give the relative effective sample size (RESS, the second line) and conditional acceptance ratio (CAR, the third line).

Tree	CRB	CRBD	TDB	TDBD	ClaDS0	ClaDS1	ClaDS2	LSBDS	BAMM
Accipitridae	-1219.1 ± 0.1	-1218.9 ± 0.1	-1220.6 ± 0.2	-1220.1 ± 0.1	-1196.7 ± 0.7	-1196.9 ± 1.1	-1196.7 ± 1.0	-1214.1 ± 3.6	-1212.3 ± 3.0
	RESS: 0.995	RESS: 0.994	RESS: 0.970	RESS: 0.987	RESS: 0.656	RESS: 0.042	RESS: 0.442	RESS: 0.034	RESS: 0.055
	CAR: 0.962	CAR: 0.956	CAR: 0.901	CAR: 0.935	CAR: 0.630	CAR: 0.331	CAR: 0.500	CAR: 0.092	CAR: 0.119
Alcedinidae	-304.4 ± 0.0	-305.5 ± 0.1	-305.6 ± 0.1	-306.0 ± 0.1	-306.9 ± 0.2	-308.9 ± 0.7	-307.7 ± 0.6	-307.5 ± 0.4	-308.6 ± 0.6
	RESS: 0.998	RESS: 0.997	RESS: 0.995	RESS: 0.996	RESS: 0.940	RESS: 0.112	RESS: 0.521	RESS: 0.857	RESS: 0.831
	CAR: 0.974	CAR: 0.967	CAR: 0.960	CAR: 0.965	CAR: 0.861	CAR: 0.564	CAR: 0.666	CAR: 0.781	CAR: 0.784
Anatinae	-587.8 ± 0.0	-586.2 ± 0.0	-587.8 ± 0.1	-586.7 ± 0.0	-576.1 ± 0.7	-576.6 ± 1.0	-575.9 ± 0.9	-581.0 ± 1.2	-579.8 ± 1.2
	RESS: 0.999	RESS: 0.998	RESS: 0.995	RESS: 0.998	RESS: 0.547	RESS: 0.041	RESS: 0.376	RESS: 0.236	RESS: 0.053
	CAR: 0.979	CAR: 0.977	CAR: 0.962	CAR: 0.972	CAR: 0.599	CAR: 0.365	CAR: 0.511	CAR: 0.393	CAR: 0.292
Caprimulgidae	-347.8 ± 0.1	-349.1 ± 0.1	-349.4 ± 0.1	-350.1 ± 0.1	-348.1 ± 0.5	-348.7 ± 0.7	-348.5 ± 0.6	-351.5 ± 1.7	-352.3 ± 3.2
	RESS: 0.997	RESS: 0.994	RESS: 0.990	RESS: 0.992	RESS: 0.775	RESS: 0.475	RESS: 0.617	RESS: 0.084	RESS: 0.004
	CAR: 0.970	CAR: 0.955	CAR: 0.942	CAR: 0.949	CAR: 0.735	CAR: 0.580	CAR: 0.638	CAR: 0.399	CAR: 0.030
Campephagidae-	-395.7 ± 0.0	-397.0 ± 0.1	-396.4 ± 0.1	-396.9 ± 0.1	-397.0 ± 0.2	-398.7 ± 0.5	-397.9 ± 0.4	-399.0 ± 0.3	-399.7 ± 0.3
	RESS: 0.998	RESS: 0.996	RESS: 0.997	RESS: 0.997	RESS: 0.944	RESS: 0.219	RESS: 0.869	RESS: 0.903	RESS: 0.888
	CAR: 0.975	CAR: 0.962	CAR: 0.967	CAR: 0.968	CAR: 0.863	CAR: 0.667	CAR: 0.794	CAR: 0.821	CAR: 0.815
Charadrii	-400.3 ± 0.1	-399.9 ± 0.1	-401.9 ± 0.1	-400.2 ± 0.1	-404.3 ± 0.3	-404.5 ± 0.7	-402.4 ± 0.7	-402.2 ± 0.4	-403.9 ± 0.9
	RESS: 0.996	RESS: 0.996	RESS: 0.983	RESS: 0.995	RESS: 0.895	RESS: 0.601	RESS: 0.541	RESS: 0.833	RESS: 0.353
	CAR: 0.966	CAR: 0.966	CAR: 0.925	CAR: 0.961	CAR: 0.812	CAR: 0.632	CAR: 0.615	CAR: 0.770	CAR: 0.532
Columbidae	-889.0 ± 0.1	-890.7 ± 0.1	-889.4 ± 0.1	-888.9 ± 0.1	-887.4 ± 0.7	-890.4 ± 1.7	-888.4 ± 1.2	-894.1 ± 0.9	-894.0 ± 4.1
	RESS: 0.997	RESS: 0.987	RESS: 0.989	RESS: 0.992	RESS: 0.606	RESS: 0.186	RESS: 0.412	RESS: 0.589	RESS: 0.185
	CAR: 0.969	CAR: 0.936	CAR: 0.940	CAR: 0.951	CAR: 0.603	CAR: 0.307	CAR: 0.459	CAR: 0.581	CAR: 0.389
Corvidae+	-1594.3 ± 0.1	-1596.8 ± 0.1	-1595.1 ± 0.1	-1596.5 ± 0.1	-1586.0 ± 0.8	-1589.0 ± 1.9	-1587.9 ± 1.3	-1600.0 ± 1.0	-1600.7 ± 3.2
	RESS: 0.996	RESS: 0.981	RESS: 0.989	RESS: 0.980	RESS: 0.499	RESS: 0.148	RESS: 0.386	RESS: 0.468	RESS: 0.276
	CAR: 0.966	CAR: 0.922	CAR: 0.939	CAR: 0.919	CAR: 0.550	CAR: 0.340	CAR: 0.431	CAR: 0.507	CAR: 0.408
Cuculidae	-881.5 ± 0.1	-883.2 ± 0.1	-883.1 ± 0.2	-883.2 ± 0.1	-882.1 ± 0.4	-884.0 ± 0.6	-882.7 ± 0.5	-886.1 ± 0.6	-889.0 ± 2.8
	RESS: 0.994	RESS: 0.982	RESS: 0.976	RESS: 0.987	RESS: 0.855	RESS: 0.698	RESS: 0.794	RESS: 0.705	RESS: 0.129
	CAR: 0.957	CAR: 0.923	CAR: 0.912	CAR: 0.937	CAR: 0.776	CAR: 0.676	CAR: 0.729	CAR: 0.671	CAR: 0.277
Emberizidae-	-737.9 ± 0.0	-740.7 ± 0.1	-725.7 ± 0.2	-727.5 ± 0.4	-731.2 ± 0.4	-735.4 ± 1.2	-732.7 ± 0.5	-744.2 ± 1.2	-733.7 ± 2.3
	RESS: 0.998	RESS: 0.985	RESS: 0.969	RESS: 0.870	RESS: 0.831	RESS: 0.508	RESS: 0.715	RESS: 0.428	RESS: 0.080
	CAR: 0.974	CAR: 0.930	CAR: 0.899	CAR: 0.783	CAR: 0.756	CAR: 0.556	CAR: 0.700	CAR: 0.472	CAR: 0.172
Estrildidae	-567.4 ± 0.0	-569.0 ± 0.1	-567.6 ± 0.1	-568.4 ± 0.1	-569.5 ± 0.7	-571.2 ± 1.1	-570.0 ± 1.1	-570.9 ± 1.0	-569.3 ± 1.5
	RESS: 0.998	RESS: 0.995	RESS: 0.998	RESS: 0.996	RESS: 0.639	RESS: 0.384	RESS: 0.394	RESS: 0.320	RESS: 0.064
	CAR: 0.977	CAR: 0.961	CAR: 0.972	CAR: 0.966	CAR: 0.629	CAR: 0.452	CAR: 0.464	CAR: 0.471	CAR: 0.199
Fringillidae+	-727.7 ± 0.0	-728.8 ± 0.1	-728.8 ± 0.1	-729.6 ± 0.1	-718.2 ± 0.6	-720.3 ± 0.6	-719.3 ± 0.7	-726.4 ± 0.9	-726.3 ± 0.9
	RESS: 0.998	RESS: 0.996	RESS: 0.995	RESS: 0.994	RESS: 0.706	RESS: 0.674	RESS: 0.659	RESS: 0.439	RESS: 0.377
	CAR: 0.974	CAR: 0.962	CAR: 0.960	CAR: 0.958	CAR: 0.697	CAR: 0.657	CAR: 0.639	CAR: 0.533	CAR: 0.495
Furnaridae	-1262.7 ± 0.0	-1265.0 ± 0.1	-1263.0 ± 0.1	-1264.7 ± 0.1	-1253.5 ± 0.8	-1256.0 ± 1.0	-1255.1 ± 0.9	-1264.1 ± 1.0	-1262.8 ± 0.9
	RESS: 0.998	RESS: 0.989	RESS: 0.996	RESS: 0.988	RESS: 0.537	RESS: 0.456	RESS: 0.494	RESS: 0.426	RESS: 0.487
	CAR: 0.974	CAR: 0.941	CAR: 0.965	CAR: 0.938	CAR: 0.594	CAR: 0.520	CAR: 0.547	CAR: 0.509	CAR: 0.527
Hirundinidae	-433.1 ± 0.0	-434.9 ± 0.1	-432.1 ± 0.1	-433.0 ± 0.1	-433.5 ± 0.3	-435.6 ± 0.6	-434.2 ± 0.5	-437.1 ± 0.5	-436.1 ± 0.8
	RESS: 0.998	RESS: 0.993	RESS: 0.997	RESS: 0.995	RESS: 0.909	RESS: 0.740	RESS: 0.735	RESS: 0.798	RESS: 0.078
	CAR: 0.974	CAR: 0.952	CAR: 0.970	CAR: 0.961	CAR: 0.826	CAR: 0.684	CAR: 0.701	CAR: 0.732	CAR: 0.454
Icteridae	-495.6 ± 0.0	-497.9 ± 0.1	-492.6 ± 0.1	-494.2 ± 0.1	-494.6 ± 0.4	-497.4 ± 0.6	-495.8 ± 0.5	-500.2 ± 0.5	-497.6 ± 0.6
	RESS: 0.999	RESS: 0.992	RESS: 0.995	RESS: 0.987	RESS: 0.863	RESS: 0.714	RESS: 0.808	RESS: 0.816	RESS: 0.664
	CAR: 0.979	CAR: 0.948	CAR: 0.962	CAR: 0.936	CAR: 0.787	CAR: 0.683	CAR: 0.744	CAR: 0.738	CAR: 0.652
Lari	-743.9 ± 0.0	-738.2 ± 0.0	-742.2 ± 0.1	-738.4 ± 0.1	-710.9 ± 0.5	-712.2 ± 1.0	-711.9 ± 0.9	-718.7 ± 1.1	-718.6 ± 1.0
	RESS: 0.998	RESS: 0.998	RESS: 0.989	RESS: 0.996	RESS: 0.783	RESS: 0.482	RESS: 0.558	RESS: 0.502	RESS: 0.536
	CAR: 0.978	CAR: 0.972	CAR: 0.942	CAR: 0.964	CAR: 0.717	CAR: 0.538	CAR: 0.570	CAR: 0.520	CAR: 0.556
Malaconotidae+	-501.2 ± 0.1	-503.3 ± 0.1	-498.0 ± 0.1	-497.8 ± 0.1	-494.9 ± 0.5	-498.2 ± 1.1	-495.7 ± 0.7	-506.4 ± 0.9	-503.6 ± 2.6
	RESS: 0.997	RESS: 0.987	RESS: 0.995	RESS: 0.994	RESS: 0.797	RESS: 0.438	RESS: 0.732	RESS: 0.458	RESS: 0.056
	CAR: 0.968	CAR: 0.935	CAR: 0.959	CAR: 0.957	CAR: 0.736	CAR: 0.499	CAR: 0.685	CAR: 0.554	CAR: 0.213
Meliphagidae+	-570.6 ± 0.1	-572.8 ± 0.1	-568.6 ± 0.1	-569.1 ± 0.1	-570.4 ± 0.5	-573.0 ± 1.1	-571.2 ± 0.9	-575.9 ± 1.0	-574.4 ± 1.5
	RESS: 0.997	RESS: 0.986	RESS: 0.994	RESS: 0.993	RESS: 0.773	RESS: 0.249	RESS: 0.413	RESS: 0.530	RESS: 0.084
	CAR: 0.968	CAR: 0.932	CAR: 0.955	CAR: 0.954	CAR: 0.723	CAR: 0.423	CAR: 0.535	CAR: 0.598	CAR: 0.296
Muscicapidae+	-1576.7 ± 0.1	-1580.3 ± 0.3	-1541.9 ± 0.2	-1543.4 ± 0.6	-1548.3 ± 0.9	-1553.9 ± 2.4	-1549.7 ± 1.4	-1586.3 ± 2.9	-1557.3 ± 8.1
	RESS: 0.996	RESS: 0.942	RESS: 0.944	RESS: 0.823	RESS: 0.390	RESS: 0.051	RESS: 0.267	RESS: 0.147	RESS: 0.016
	CAR: 0.964	CAR: 0.860	CAR: 0.864	CAR: 0.736	CAR: 0.501	CAR: 0.178	CAR: 0.370	CAR: 0.209	CAR: 0.043
Paridae+	-327.2 ± 0.0	-328.8 ± 0.1	-327.8 ± 0.1	-327.8 ± 0.1	-319.1 ± 0.6	-321.5 ± 0.8	-320.0 ± 0.7	-331.0 ± 1.0	-327.0 ± 3.2
	RESS: 0.998	RESS: 0.993	RESS: 0.994	RESS: 0.996	RESS: 0.536	RESS: 0.534	RESS: 0.626	RESS: 0.437	RESS: 0.038
	CAR: 0.972	CAR: 0.953	CAR: 0.956	CAR: 0.964	CAR: 0.649	CAR: 0.579	CAR: 0.637	CAR: 0.608	CAR: 0.102

Table 7: (continued)

Tree	CRB	CRBD	TDB	TDBD	ClaDS0	ClaDS1	ClaDS2	LSBDS	BAMM
Parulidae+	-620.5 ± 0.0 RESS: 0.998 CAR: 0.978	-622.7 ± 0.1 RESS: 0.992 CAR: 0.950	-619.7 ± 0.1 RESS: 0.997 CAR: 0.969	-621.3 ± 0.1 RESS: 0.993 CAR: 0.953	-600.3 ± 0.9 RESS: 0.468 CAR: 0.539	-603.1 ± 1.0 RESS: 0.421 CAR: 0.498	-601.2 ± 1.0 RESS: 0.320 CAR: 0.476	-622.6 ± 1.6 RESS: 0.227 CAR: 0.362	-615.1 ± 2.8 RESS: 0.030 CAR: 0.073
	-808.7 ± 0.0 RESS: 0.998 CAR: 0.972	-809.5 ± 0.1 RESS: 0.996 CAR: 0.963	-809.9 ± 0.1 RESS: 0.993 CAR: 0.954	-809.5 ± 0.1 RESS: 0.996 CAR: 0.964	-810.7 ± 0.5 RESS: 0.813 CAR: 0.746	-811.0 ± 0.7 RESS: 0.558 CAR: 0.611	-810.4 ± 0.8 RESS: 0.460 CAR: 0.564	-812.0 ± 0.7 RESS: 0.573 CAR: 0.613	-812.9 ± 1.1 RESS: 0.125 CAR: 0.400
Picidae	-830.7 ± 0.0 RESS: 0.998 CAR: 0.974	-832.8 ± 0.1 RESS: 0.991 CAR: 0.945	-831.8 ± 0.1 RESS: 0.995 CAR: 0.959	-833.4 ± 0.1 RESS: 0.987 CAR: 0.934	-828.5 ± 0.8 RESS: 0.547 CAR: 0.597	-830.2 ± 1.1 RESS: 0.334 CAR: 0.444	-830.6 ± 1.5 RESS: 0.190 CAR: 0.339	-835.4 ± 1.5 RESS: 0.173 CAR: 0.291	-835.7 ± 2.7 RESS: 0.028 CAR: 0.068
	-686.0 ± 0.1 RESS: 0.997 CAR: 0.967	-684.0 ± 0.1 RESS: 0.996 CAR: 0.965	-686.5 ± 0.2 RESS: 0.972 CAR: 0.905	-684.8 ± 0.1 RESS: 0.993 CAR: 0.954	-680.8 ± 0.7 RESS: 0.638 CAR: 0.628	-681.9 ± 0.9 RESS: 0.444 CAR: 0.559	-681.0 ± 0.9 RESS: 0.486 CAR: 0.535	-685.4 ± 1.0 RESS: 0.324 CAR: 0.447	-687.1 ± 1.4 RESS: 0.058 CAR: 0.299
Psittacidae1	-690.7 ± 0.1 RESS: 0.997 CAR: 0.972	-688.8 ± 0.1 RESS: 0.997 CAR: 0.969	-690.9 ± 0.1 RESS: 0.987 CAR: 0.934	-689.2 ± 0.1 RESS: 0.995 CAR: 0.962	-691.9 ± 0.6 RESS: 0.713 CAR: 0.685	-692.5 ± 1.0 RESS: 0.426 CAR: 0.511	-691.5 ± 0.9 RESS: 0.339 CAR: 0.493	-691.4 ± 0.7 RESS: 0.636 CAR: 0.633	-693.0 ± 2.0 RESS: 0.232 CAR: 0.419
	-729.5 ± 0.0 RESS: 0.998 CAR: 0.972	-730.3 ± 0.1 RESS: 0.995 CAR: 0.962	-730.7 ± 0.1 RESS: 0.990 CAR: 0.944	-731.3 ± 0.1 RESS: 0.992 CAR: 0.950	-726.6 ± 0.5 RESS: 0.772 CAR: 0.715	-727.8 ± 0.7 RESS: 0.194 CAR: 0.561	-727.3 ± 0.7 RESS: 0.529 CAR: 0.585	-731.5 ± 0.8 RESS: 0.436 CAR: 0.540	-732.3 ± 0.7 RESS: 0.595 CAR: 0.643
Pycnonotidae+	-589.9 ± 0.1 RESS: 0.997 CAR: 0.970	-592.3 ± 0.1 RESS: 0.983 CAR: 0.926	-584.0 ± 0.1 RESS: 0.995 CAR: 0.958	-584.9 ± 0.1 RESS: 0.991 CAR: 0.947	-585.9 ± 0.5 RESS: 0.803 CAR: 0.740	-588.9 ± 0.9 RESS: 0.457 CAR: 0.545	-586.9 ± 0.7 RESS: 0.638 CAR: 0.626	-595.1 ± 0.6 RESS: 0.772 CAR: 0.706	-589.8 ± 1.9 RESS: 0.482 CAR: 0.542
	-475.0 ± 0.0 RESS: 0.998 CAR: 0.974	-475.0 ± 0.1 RESS: 0.998 CAR: 0.972	-475.9 ± 0.1 RESS: 0.993 CAR: 0.952	-475.7 ± 0.1 RESS: 0.996 CAR: 0.966	-476.8 ± 0.5 RESS: 0.795 CAR: 0.736	-478.3 ± 0.8 RESS: 0.480 CAR: 0.584	-477.4 ± 0.7 RESS: 0.493 CAR: 0.606	-477.0 ± 0.5 RESS: 0.811 CAR: 0.750	-478.2 ± 0.6 RESS: 0.483 CAR: 0.633
Strigidae	-645.2 ± 0.1 RESS: 0.997 CAR: 0.968	-646.4 ± 0.1 RESS: 0.993 CAR: 0.954	-646.8 ± 0.1 RESS: 0.987 CAR: 0.935	-647.7 ± 0.1 RESS: 0.984 CAR: 0.928	-647.5 ± 0.6 RESS: 0.689 CAR: 0.677	-649.8 ± 1.0 RESS: 0.501 CAR: 0.531	-649.1 ± 0.9 RESS: 0.468 CAR: 0.517	-649.1 ± 0.8 RESS: 0.560 CAR: 0.589	-650.9 ± 1.0 RESS: 0.464 CAR: 0.558
	-794.2 ± 0.1 RESS: 0.997 CAR: 0.971	-796.6 ± 0.1 RESS: 0.987 CAR: 0.935	-792.9 ± 0.1 RESS: 0.995 CAR: 0.961	-794.2 ± 0.1 RESS: 0.992 CAR: 0.949	-793.2 ± 0.6 RESS: 0.670 CAR: 0.661	-795.6 ± 1.0 RESS: 0.302 CAR: 0.481	-794.5 ± 1.0 RESS: 0.408 CAR: 0.520	-799.2 ± 0.7 RESS: 0.668 CAR: 0.652	-797.9 ± 0.9 RESS: 0.426 CAR: 0.520
Sylviidae1+	-453.3 ± 0.0 RESS: 0.998 CAR: 0.975	-454.1 ± 0.1 RESS: 0.997 CAR: 0.971	-454.4 ± 0.1 RESS: 0.995 CAR: 0.962	-454.6 ± 0.1 RESS: 0.996 CAR: 0.966	-442.9 ± 0.5 RESS: 0.787 CAR: 0.721	-445.0 ± 0.7 RESS: 0.519 CAR: 0.620	-444.2 ± 0.7 RESS: 0.605 CAR: 0.616	-451.0 ± 0.7 RESS: 0.571 CAR: 0.612	-451.5 ± 0.9 RESS: 0.357 CAR: 0.569
	-540.6 ± 0.0 RESS: 0.998 CAR: 0.975	-542.1 ± 0.1 RESS: 0.994 CAR: 0.958	-541.3 ± 0.1 RESS: 0.997 CAR: 0.967	-541.7 ± 0.1 RESS: 0.996 CAR: 0.966	-537.4 ± 0.6 RESS: 0.698 CAR: 0.678	-539.4 ± 0.7 RESS: 0.634 CAR: 0.631	-538.4 ± 0.7 RESS: 0.603 CAR: 0.610	-544.2 ± 0.6 RESS: 0.659 CAR: 0.664	-544.2 ± 1.1 RESS: 0.195 CAR: 0.423
Thamnophilidae	-1061.2 ± 0.1 RESS: 0.997 CAR: 0.970	-1064.2 ± 0.2 RESS: 0.974 CAR: 0.909	-1049.0 ± 0.1 RESS: 0.989 CAR: 0.940	-1050.5 ± 0.2 RESS: 0.970 CAR: 0.901	-1046.9 ± 0.9 RESS: 0.513 CAR: 0.542	-1050.2 ± 2.3 RESS: 0.140 CAR: 0.306	-1048.6 ± 1.3 RESS: 0.206 CAR: 0.365	-1067.7 ± 1.1 RESS: 0.476 CAR: 0.511	-1056.6 ± 2.4 RESS: 0.002 CAR: 0.004
	-935.9 ± 0.0 RESS: 0.998 CAR: 0.976	-938.3 ± 0.1 RESS: 0.989 CAR: 0.940	-931.3 ± 0.1 RESS: 0.993 CAR: 0.951	-932.2 ± 0.2 RESS: 0.977 CAR: 0.913	-919.5 ± 1.0 RESS: 0.309 CAR: 0.476	-922.8 ± 1.2 RESS: 0.180 CAR: 0.391	-920.1 ± 1.0 RESS: 0.406 CAR: 0.484	-935.9 ± 0.8 RESS: 0.559 CAR: 0.594	-926.0 ± 1.1 RESS: 0.264 CAR: 0.423
Thraupidae1+	-807.6 ± 0.0 RESS: 0.998 CAR: 0.976	-810.1 ± 0.1 RESS: 0.988 CAR: 0.938	-801.6 ± 0.1 RESS: 0.990 CAR: 0.944	-803.1 ± 0.2 RESS: 0.970 CAR: 0.901	-791.0 ± 0.6 RESS: 0.730 CAR: 0.682	-794.1 ± 1.0 RESS: 0.471 CAR: 0.528	-791.7 ± 0.7 RESS: 0.558 CAR: 0.595	-807.6 ± 1.8 RESS: 0.311 CAR: 0.522	-798.7 ± 1.6 RESS: 0.109 CAR: 0.239
	-1120.2 ± 0.0 RESS: 0.998 CAR: 0.974	-1121.0 ± 0.1 RESS: 0.996 CAR: 0.963	-1121.4 ± 0.1 RESS: 0.995 CAR: 0.960	-1121.4 ± 0.1 RESS: 0.995 CAR: 0.958	-1082.9 ± 1.5 RESS: 0.205 CAR: 0.332	-1084.8 ± 1.6 RESS: 0.161 CAR: 0.295	-1084.3 ± 1.8 RESS: 0.076 CAR: 0.206	-1096.5 ± 2.2 RESS: 0.131 CAR: 0.207	-1095.5 ± 3.2 RESS: 0.013 CAR: 0.030
Trochilidae	-1567.5 ± 0.1 RESS: 0.997 CAR: 0.968	-1569.7 ± 0.1 RESS: 0.983 CAR: 0.926	-1567.9 ± 0.1 RESS: 0.992 CAR: 0.948	-1569.0 ± 0.1 RESS: 0.986 CAR: 0.933	-1562.8 ± 0.9 RESS: 0.483 CAR: 0.534	-1565.4 ± 1.3 RESS: 0.363 CAR: 0.435	-1563.9 ± 1.2 RESS: 0.366 CAR: 0.439	-1573.0 ± 3.8 RESS: 0.273 CAR: 0.378	-1573.0 ± 3.7 RESS: 0.097 CAR: 0.190
	-545.7 ± 0.0 RESS: 0.998 CAR: 0.973	-547.8 ± 0.1 RESS: 0.989 CAR: 0.941	-546.0 ± 0.1 RESS: 0.996 CAR: 0.963	-547.4 ± 0.1 RESS: 0.991 CAR: 0.948	-547.8 ± 0.4 RESS: 0.846 CAR: 0.773	-550.5 ± 0.7 RESS: 0.643 CAR: 0.621	-549.3 ± 0.6 RESS: 0.742 CAR: 0.693	-550.3 ± 0.6 RESS: 0.736 CAR: 0.681	-550.7 ± 1.0 RESS: 0.346 CAR: 0.515
Turdidae+	-830.1 ± 0.1 RESS: 0.997 CAR: 0.970	-833.0 ± 0.2 RESS: 0.975 CAR: 0.910	-821.3 ± 0.1 RESS: 0.992 CAR: 0.949	-822.7 ± 0.2 RESS: 0.978 CAR: 0.915	-813.0 ± 0.9 RESS: 0.508 CAR: 0.550	-817.4 ± 1.5 RESS: 0.299 CAR: 0.373	-814.4 ± 1.0 RESS: 0.461 CAR: 0.498	-836.0 ± 0.8 RESS: 0.578 CAR: 0.594	-826.4 ± 2.2 RESS: 0.014 CAR: 0.043
	-2273.6 ± 0.1 RESS: 0.995 CAR: 0.961	-2276.0 ± 0.2 RESS: 0.971 CAR: 0.902	-2274.1 ± 0.1 RESS: 0.985 CAR: 0.931	-2275.1 ± 0.2 RESS: 0.978 CAR: 0.915	-2262.0 ± 1.0 RESS: 0.334 CAR: 0.489	-2262.7 ± 1.6 RESS: 0.176 CAR: 0.300	-2260.9 ± 1.3 RESS: 0.275 CAR: 0.387	-2278.2 ± 8.8 RESS: 0.096 CAR: 0.171	-2281.7 ± 11.4 RESS: 0.004 CAR: 0.006

Table 8 Median execution time in Birch in seconds for each tree and model. See text for details.

Tree	CRB	CRBD	TDB	TDBD	ClaDS0	ClaDS1	ClaDS2	LSBDS	BAMM
Accipitridae	14	27	30	35	400	460	606	860	2097
Alcedinidae	4	8	8	11	77	95	214	154	792
Anatinae	8	17	19	22	222	236	631	226	1719
Caprimulgidae	4	9	10	11	96	115	209	289	958
Campephagidae-	5	11	12	14	107	120	221	124	966
Charadrii	5	11	11	13	109	134	571	328	890
Columbidae	10	20	23	27	258	295	475	310	1918
Corvidae+	18	36	41	47	570	636	820	415	2852
Cuculidae	10	19	22	25	241	271	380	441	1730
Emberizidae-	10	19	23	25	234	260	450	222	1669
Estrildidae	8	16	17	20	179	213	613	200	1344
Fringillidae+	10	18	21	25	212	233	287	209	1329
Furnariidae	15	30	35	41	463	520	625	243	1941
Hirundinidae	6	12	13	15	111	129	277	180	942
Icteridae	7	15	17	20	152	178	334	159	1077
Lari	9	19	22	25	228	238	477	295	1295
Malaconotidae+	6	12	14	16	131	155	512	813	1091
Meliphagidae+	7	14	16	18	150	173	326	257	1140
Muscicapidae-+	18	36	40	47	576	640	909	414	2941
Paridae+	4	8	9	11	84	98	216	429	677
Parulidae+	8	18	20	24	210	234	366	257	1527
Phasianidae	10	21	25	26	245	267	412	230	1348
Picidae	10	21	24	28	269	279	456	295	1611
Procellariidae	8	16	18	20	193	231	386	267	1351
Psittacidae1	8	17	19	22	205	219	477	316	1237
Psittacidae2	9	18	21	22	216	237	360	220	1425
Pycnonotidae+	7	14	17	19	144	169	243	182	1261
Ramphastidae	6	13	15	15	122	138	299	233	851
Strigidae	8	15	18	20	170	204	392	232	1198
Sturnidae+	9	19	21	27	247	251	352	237	1538
Sylviidae1+	6	10	14	16	114	141	276	279	864
Sylviidae2+	7	14	16	17	152	158	315	287	883
Thamnophilidae	12	24	28	33	327	418	524	415	1802
Thraupidae1+	12	27	27	34	314	363	428	224	1673
Thraupidae2+	11	23	24	30	327	323	723	277	1753
Timaliidae+	14	27	31	32	366	411	481	262	1864
Trochilidae	18	33	41	47	578	657	821	432	2699
Troglodytidae+	7	13	16	18	149	171	289	182	1193
Turdidae+	10	21	23	27	251	284	426	223	1447
Tyrannidae+	23	48	55	63	963	1022	2073	2064	3250

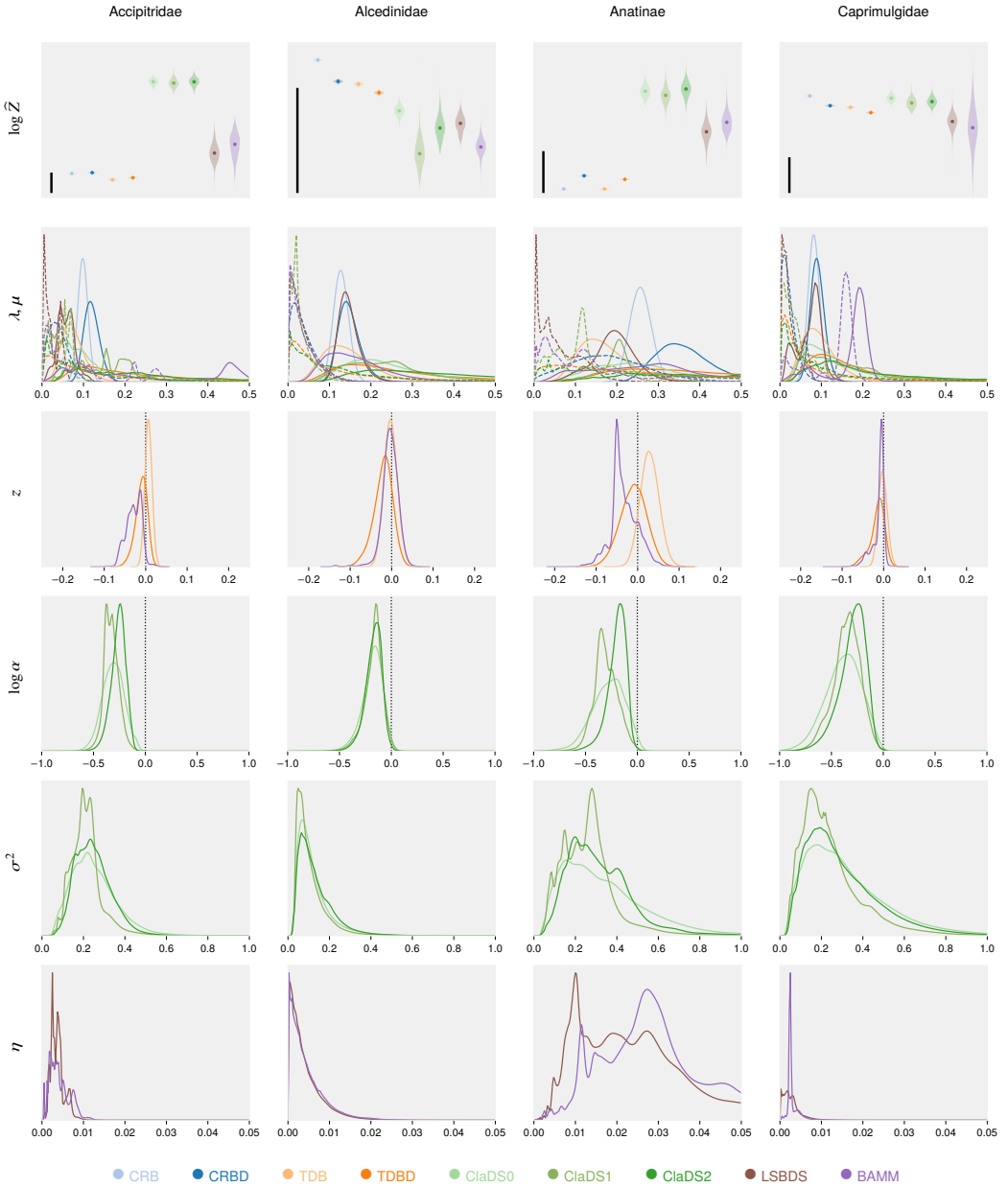


Fig. 13 Normalization constants and parameter estimates for Accipitridae, Alcedinidae, Anatinae, Caprimulgidae.

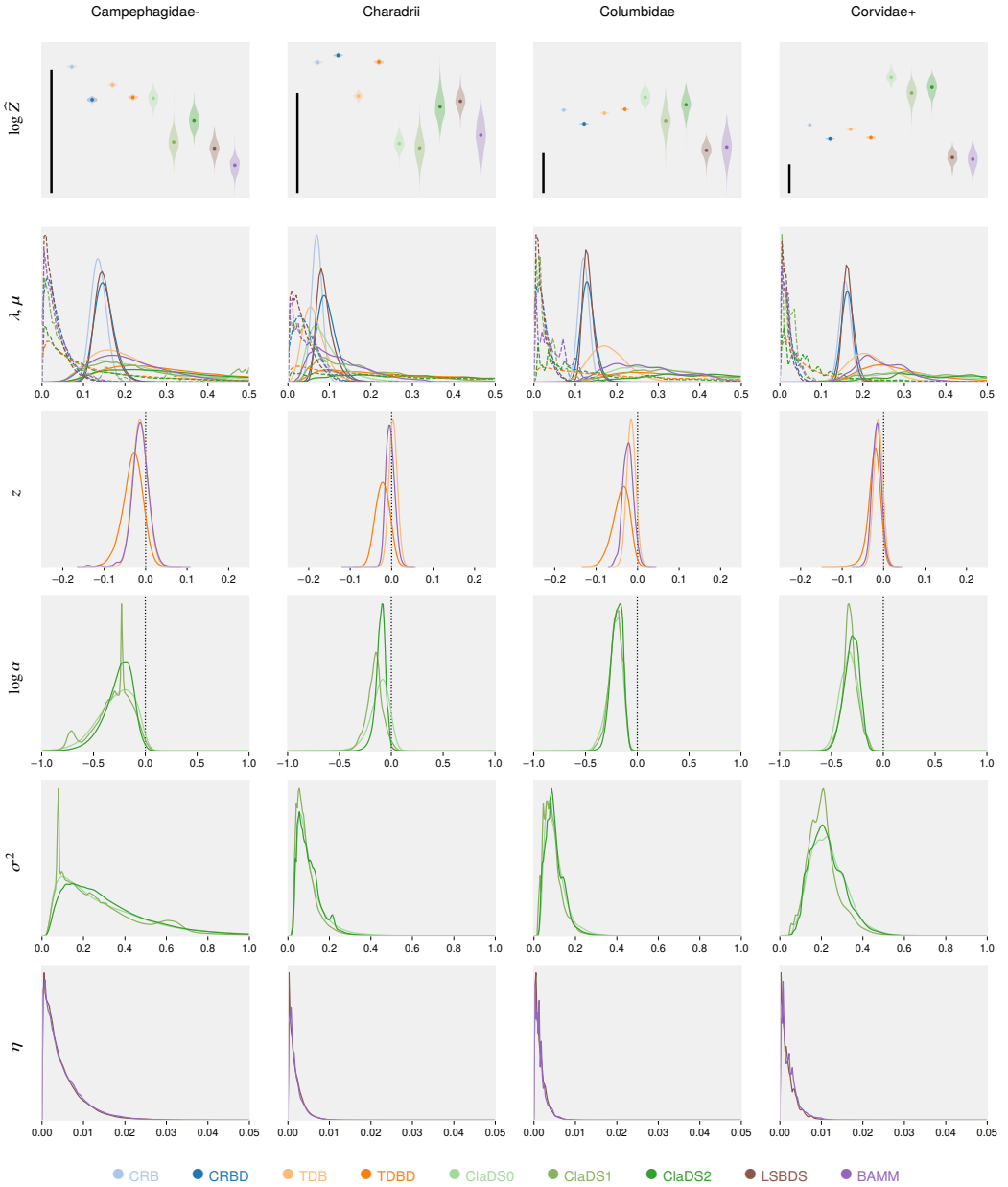


Fig. 14 Normalization constants and parameter estimates for Campephagidae-, Charadrii, Columbidae, Corvidae+.

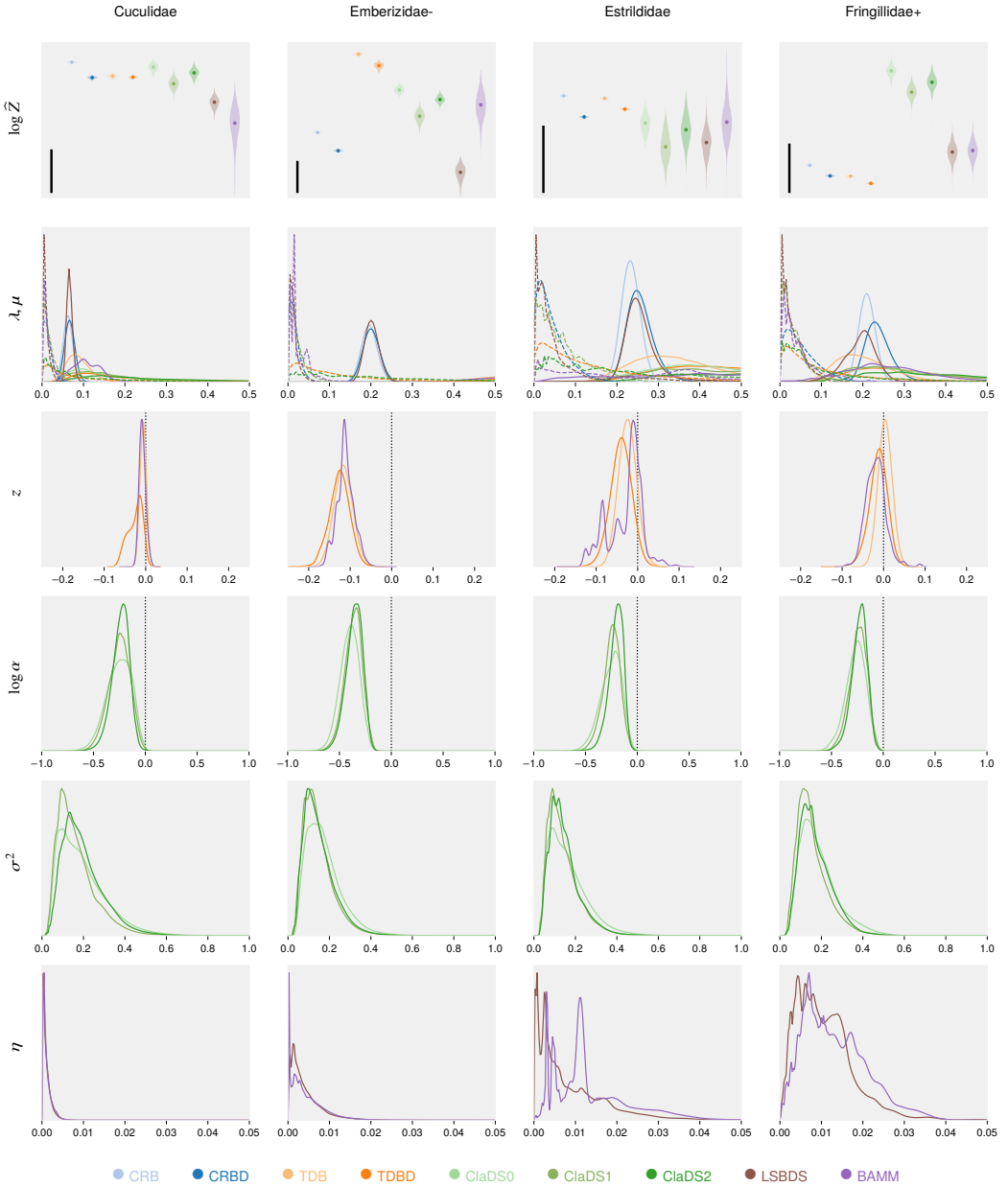


Fig. 15 Normalization constants and parameter estimates for Cuculidae, Emberizidae-, Estrildae, Fringillidae+.

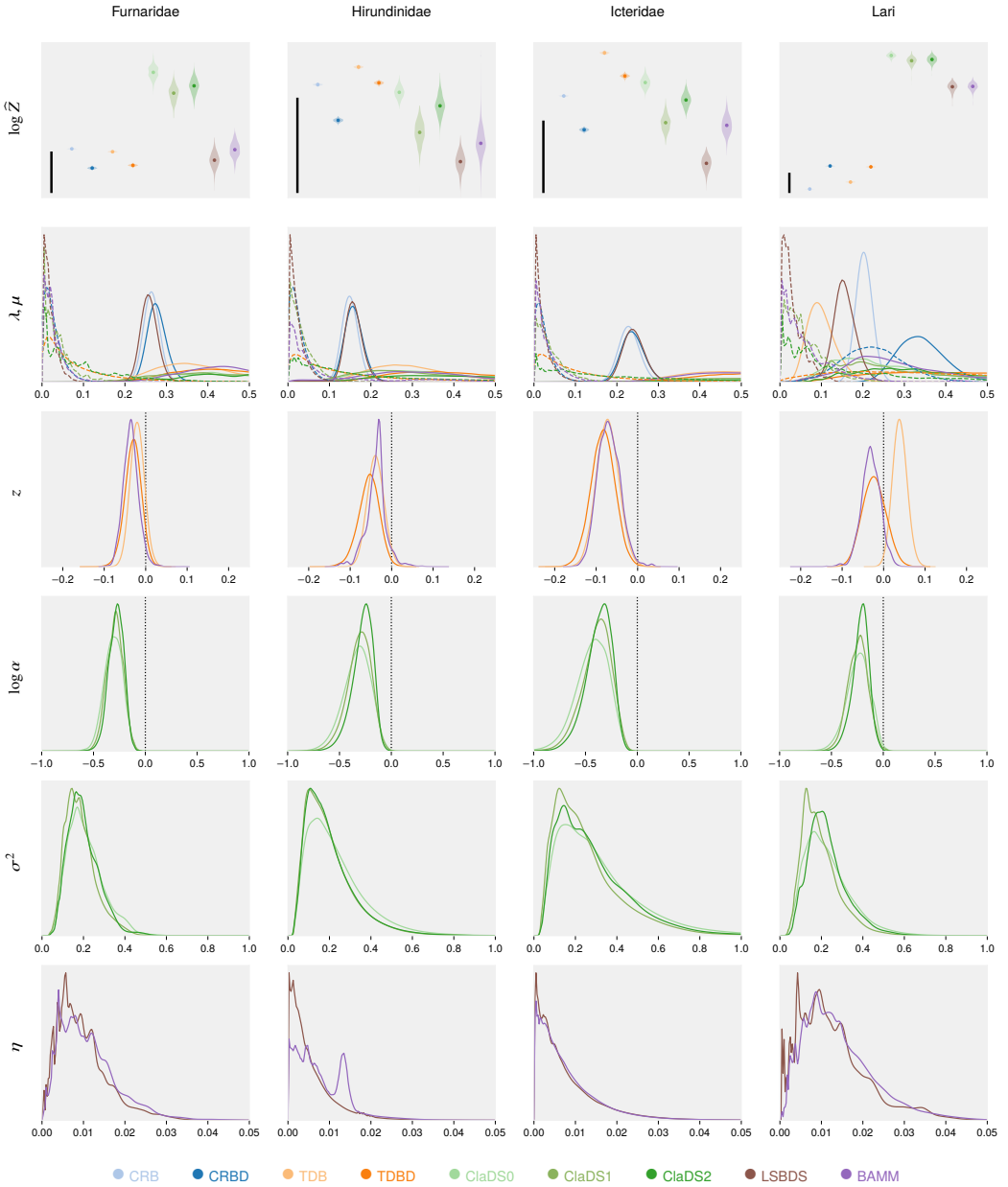


Fig. 16 Normalization constants and parameter estimates for Furnariidae, Hirundinidae, Icteridae, Lari.

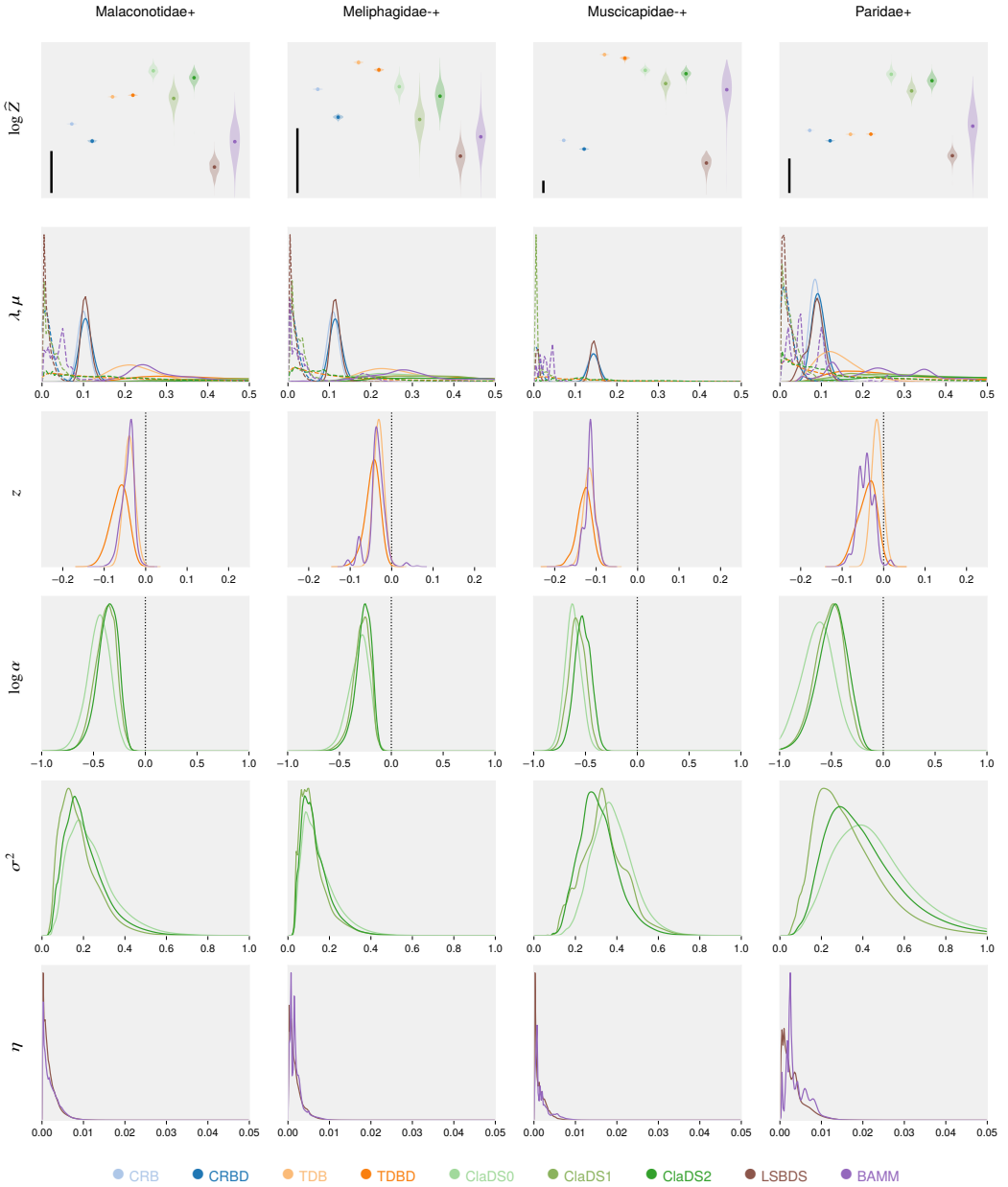


Fig. 17 Normalization constants and parameter estimates for Malaconotidae+, Meliphagidae+, Muscipidae+, Paridae+.

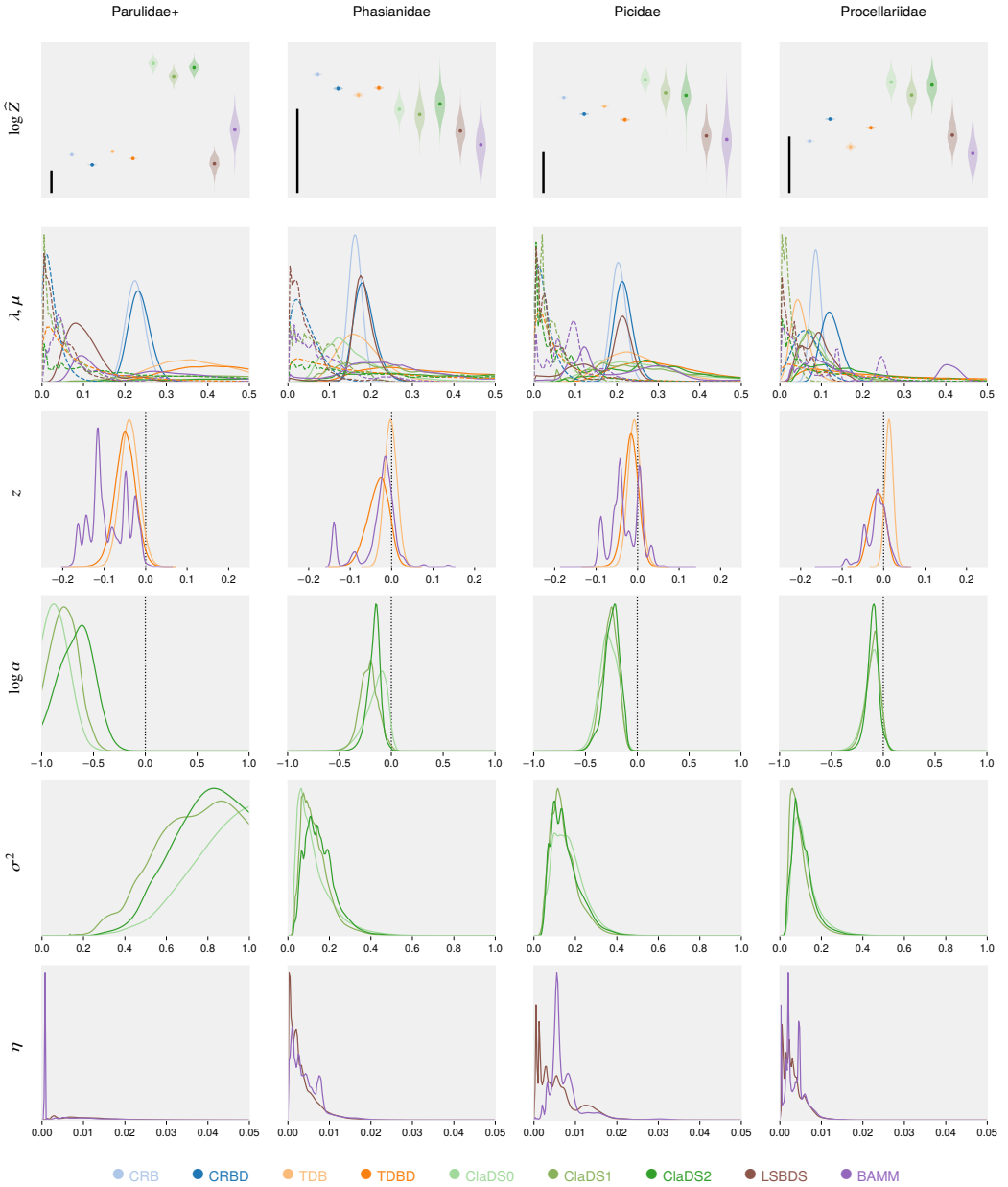


Fig. 18 Normalization constants and parameter estimates for Parulidae+, Phasianidae, Picidae, Procellariidae.

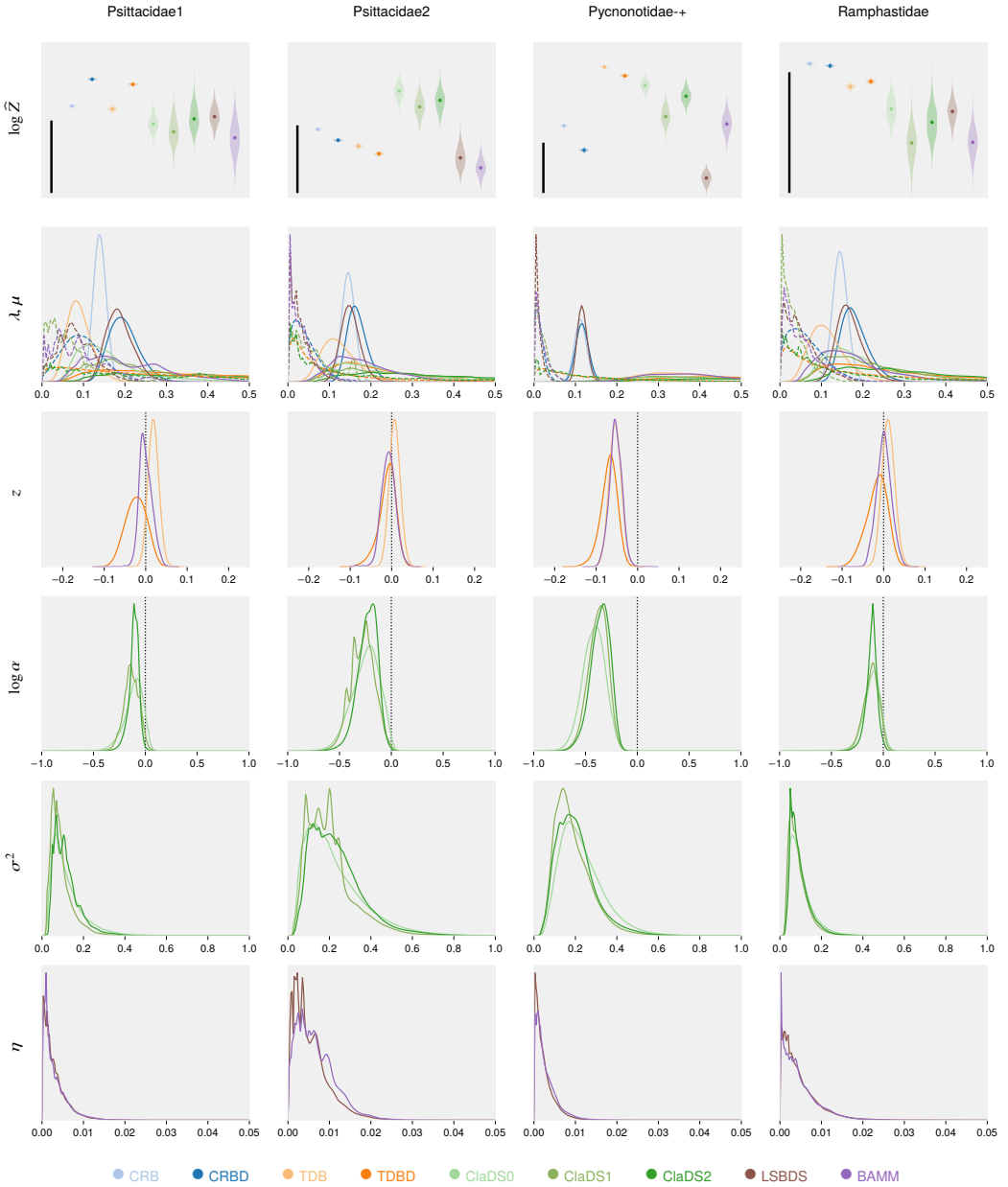


Fig. 19 Normalization constants and parameter estimates for Psittacidae1, Psittacidae2, Pycnonotidae+, Ramphastidae.

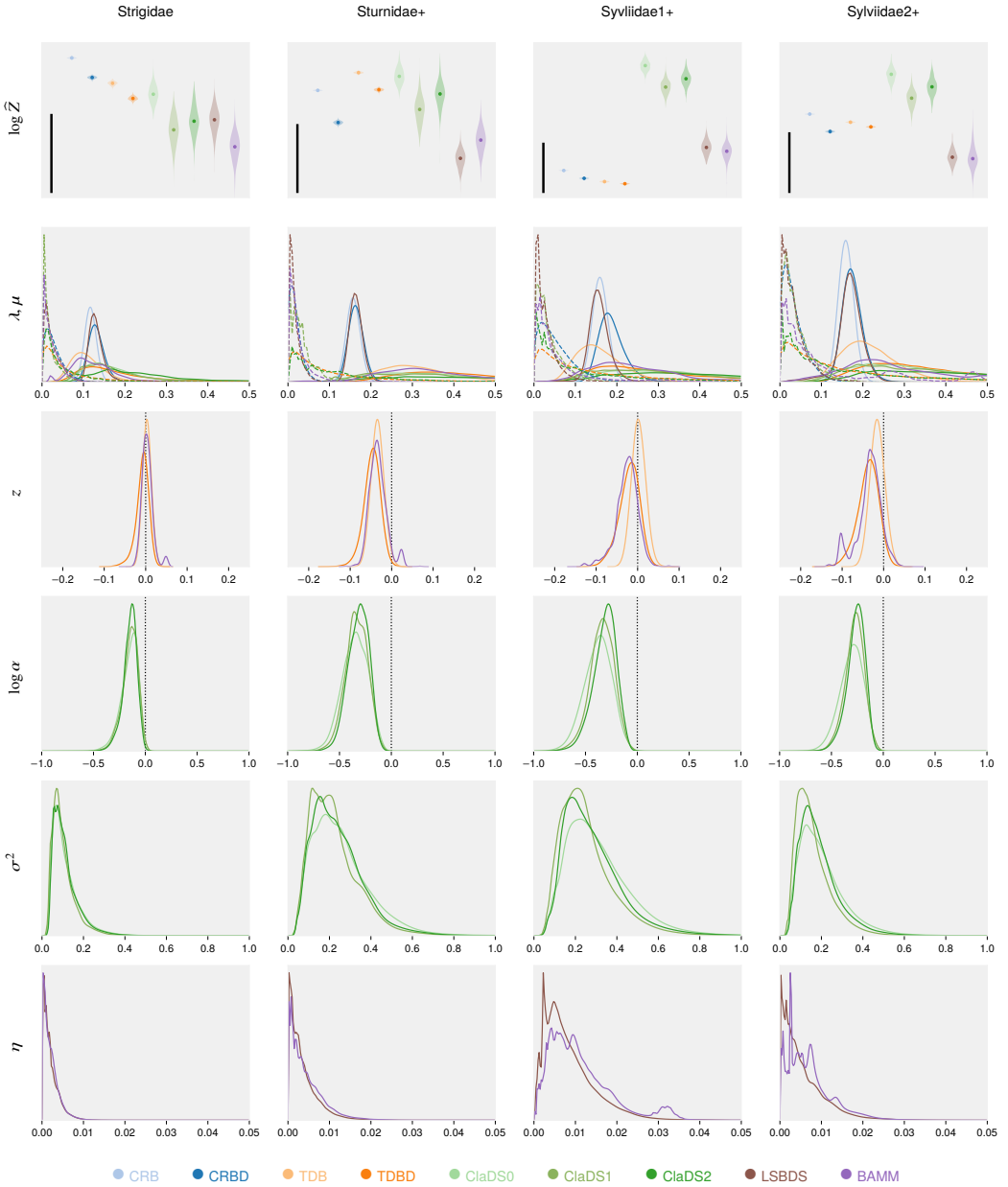


Fig. 20 Normalization constants and parameter estimates for Strigidae, Sturnidae+, Sylviidae1+, Sylviidae2+.

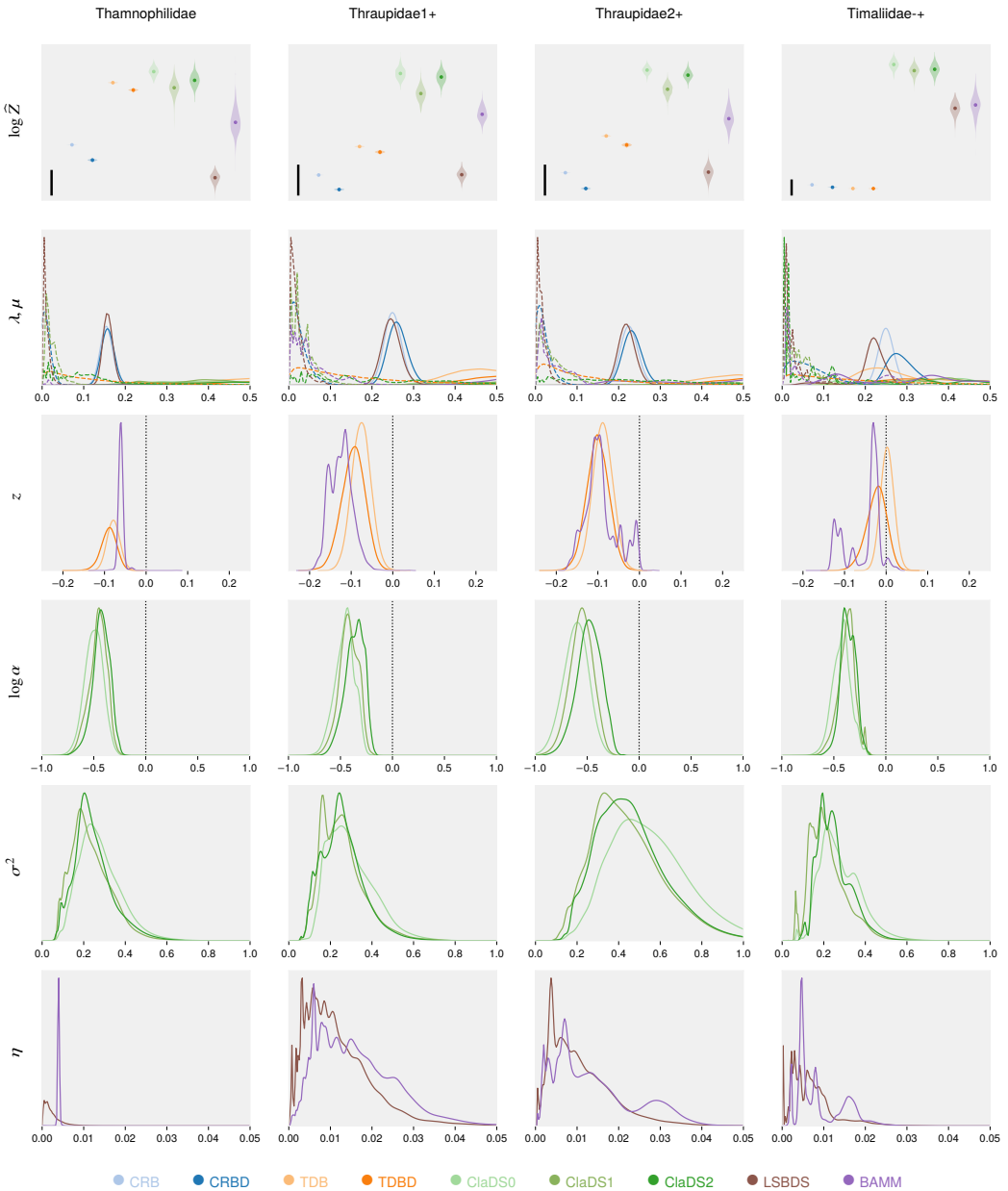


Fig. 21 Normalization constants and parameter estimates for Thamnophilidae, Thraupidae1+, Thraupidae2+, Timaliidae+.

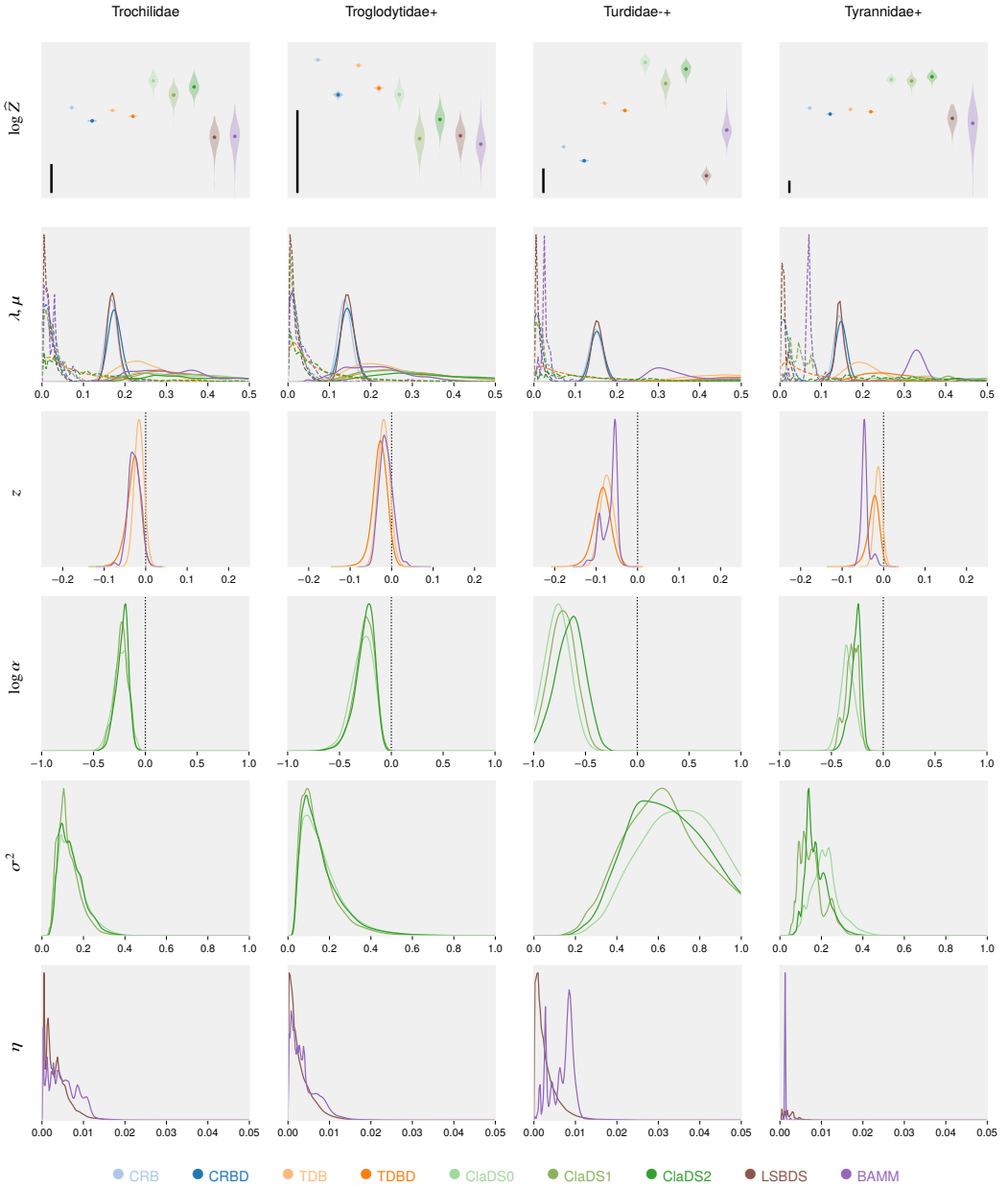


Fig. 22 Normalization constants and parameter estimates for Trochilidae, Troglodytidae+, Turdidae+, Tyrannidae+.

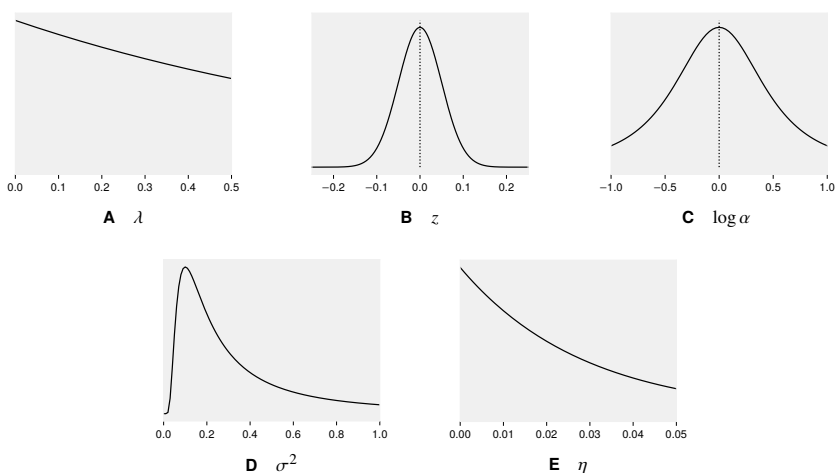


Fig. 23 Prior distributions plotted for the same region of parameter space used for the posterior distributions in Supplementary Figures 13–22.

all trees bear a clear mark of lineage-specific or, at least, slowing (TDB(D)) diversification. The age of the tree appears to be positively related to the adequacy of simple diversification models. Of the ten youngest trees, only two (Estrildidae+ and Icteridae; Supplementary Figures 15 and 16, respectively) lack strong support for lineage-specific or slowing diversification, while this is fairly common among the oldest trees.

These patterns appear to be best explained by the density of branching events in the reconstructed tree. The more branching events there are per unit time, the more likely it is that the evolutionary process has left signs of density-dependent or lineage-specific diversification. These fluctuations in diversification rates may tend to even out over longer time scales, as old and species-poor trees are often adequately explained by simple models. However, there are clear exceptions. For instance, the Paridae+ (Supplementary Figure 17) shows clear evidence of lineage-specific diversification, despite being an old group (40.9 Ma) with relatively few species (55).

Overall, our results clearly illustrate that Bayes factors are inherently conservative, preferring simpler models unless the signal in the data is sufficiently strong to decisively reject them. While this could be considered a reasonable feature, some caution is nevertheless needed when interpreting the outcome of the model comparison experiment. In particular, the fact that very simple models seem adequate for so many of the bird clades appears to largely reflect the lack of (sufficiently strong) evidence and should not be interpreted as evidence of absence. Many of the trees analysed here (and elsewhere) are too small or not informative enough to allow for a non-trivial outcome.

The degree to which Bayes factors are conservative is dependent on the prior distributions used for the additional parameters of the more complex models. More diffuse priors automatically result in higher penalties in the model comparison—and this even if the posterior distribution itself is not impacted or only marginally impacted. Often, this is not a major problem, for instance when comparing models of sequence evolution, where the signal contributed by the sequence data easily overwhelms the penalty induced by diffuse priors. Here, in contrast, the empirical signal contributed by phylogenetic trees of surviving lineages about the underlying diversification process is somewhat weaker, making the relative impact of the prior on the outcome of Bayesian model tests more substantial.

Whether our priors strike a reasonable balance between simple and complex models is, of course, open to discussion. We note, however, that our priors for the ClaDS models are less conservative than the ones proposed originally for these models³⁰. Thus, our priors penalize the ClaDS models less than would otherwise have been the case. We also want to re-emphasize that, to allow fair model comparisons, we chose priors on analogous model parameters that were similar, if not identical, across models.

An alternative to Bayesian model comparison is to focus on model adequacy, that is, the extent to which the models are consistent with the data. A popular approach to assess model adequacy is to use posterior predictive checks, but this requires the specification of an appropriate discrepancy measure⁴⁸. A general criterion of model adequacy that avoids this difficulty is the recently introduced data consistency criterion⁴⁹. However, we refrain from pursuing this topic further here.

10.2 Robustness of complex models

An important result that emerges from our analyses, and that we want to emphasize, is the robustness of complex diversification models. Even when Bayes factors indicate that simple models are adequate, the more sophisticated models often give consistent estimates for the additional model parameters. Good examples are provided by the posterior estimates for σ^2 , describing the rate of gradual, lineage-specific change in diversification rates in the ClaDS models, and η , denoting the rate of punctuated change in the LSBDS and BAMM models; both of these parameters are usually estimated to be close to 0 when simple models appear adequate. Similarly, the parameters related to potential density-dependent effects (z for TDB(D) and BAMM, and $\log \alpha$ for

ClaDS) are often close to 0 when the CRB(D) models have the best marginal likelihoods. Furthermore, no-extinction models, such as ClaDS0, often have higher marginal likelihoods than their counterparts that accommodate extinction. However, in these cases, the more complex models almost always estimate extinction or turnover rates that are close to 0. This usually occurs with very little impact on the estimation of other parameters, as is well illustrated by the very similar posterior distributions obtained across the ClaDS model series, despite the fact that ClaDS0 often has (slightly) better marginal likelihood than the more complex variants.

If the results are scrutinized, one discovers that the advanced diversification models actually appear to pick up weak but consistent signal for more complex patterns even when they are not favored by the model tests. For instance, when posterior estimates of $\log \alpha$ or z are significantly different from 0 in these cases, the estimates always suggest slowing diversification rates, and the models that accommodate such variation over time tend to be the ones with the best model likelihoods, even if they are only marginally better than the constant-rate models. Taken together, these observations suggest that the more complex models might in fact be generally more adequate than the simpler ones. The risk of obtaining erroneous or misleading inference under more complex models appears to be low, at least in comparisons among nested models with similar dimensionality.

10.3 Slowing diversification rates

The strongest signal across bird clades in our analyses is undoubtedly the support for slowing diversification rates. This is seen already in the model comparisons but perhaps more clearly in the posterior estimates of $\log \alpha$ in the ClaDS models, and z in the TDB(D) and BAMM models (Supplementary Figures 13–22). The estimates are almost universally below 0, indicating decelerating rates, and usually significantly so (more than 95% of the credible interval on negative values). Nowhere is the signal more evident than in the four bird clades where the models that only account for changing diversification rates over time—the TDB(D) models—come out distinctly ahead of all others in the model comparison (Emberizidae-, Meliphagidae-+, Muscicapidae-+ and Pycnonotidae-+). In two of those cases (Emberizidae- and Muscicapidae-+; Supplementary Figures 15 and 17, respectively), the Bayes factors even provide strong evidence in favor of TDB(D) over all other models.

Diversification rates that slow down over time are usually attributed to competition for limited resources or niches^{50,51,52}. Alternative explanations that have been proposed include: (1) subdivision of geographic ranges at speciation; (2) speciation bursts driven by environmental or geological change; (3) failure to keep pace with environmental change; and (4) protracted speciation (related to the diversified sampling bias, see below)⁵³. It might be possible to tease apart some of these factors by developing more sophisticated diversification models within the PPL framework, but this is outside the scope of the current paper. Regardless of the causes, it is clear that there is a strong signature of slowing diversification rates in the bird clades, and that it is important to account for this in diversification models.

10.4 Gradual change, punctuated change or both?

Unsurprisingly, there is also clear evidence of variation across lineages in diversification rates. Of the 40 bird trees, Bayes factors strongly favor models accommodating lineage-specific effects over simpler ones in 15 cases. Even in the remaining cases, there is often some support for lineage-specific variation in diversification rates, as indicated by posterior estimates of model parameters.

The ClaDS models consistently explain this variation in diversification rates better than the LSBDS and BAMM models. In fact, there are only three groups for which the LSBDS and BAMM models are strongly favored over the corresponding simple models: Anatinae, Lari, and Timaliidae-+ (Supplementary Figures 13, 16 and 21, respectively). The BAMM model also does comparatively well on the Thraupidae+ tree (Supplementary Figure 21). As expected, these trees are also associated with posterior estimates of η that differ substantially from 0. However, even for these trees, where BAMM and LSBDS detect major shifts in diversification rates, the ClaDS models provide a better fit to the data.

We may conclude that lineage-specific differences in diversification rates are better explained by slow, gradual changes, which accumulate over time, than by a few events that drastically alter the rates. One possible explanation for this is that the punctuated models (BAMM and LSBDS) draw the new λ and μ (and z for BAMM) values from diffuse priors at process switching events. This means that they carry heavy penalties in Bayes factor comparisons; the more switches there are, the heavier the penalty against these models. An interesting difference between the punctuated models and the gradual models is that the former allow both λ and μ to vary over the tree, while only λ is modulated over the tree in the latter. Could this be the explanation for the gradual models outperforming the punctuated models? We tested this by modifying LSBDS and BAMM such that they assumed a constant turnover rate ($\epsilon = \mu/\lambda$), as in ClaDS2, and only varied λ (and z for BAMM) at switching points. We then re-computed the normalization constants for Lari, one of the clades with the strongest evidence for major shifts in diversification rates. The normalization constants of the punctuated models did not improve noticeably due to this modification (results not shown). This finding suggests that the strong evidence in favor of gradual over punctuated change is not due simply to the punctuated models postulating changes in extinction rates that are not supported by the data.

A fascinating question is whether there remains any evidence for occasional major shifts in diversification rates if one first adequately accounts for the strong underlying signal of slow and gradual change. This can now be examined by extending the PPL framework we present here to diversification models that combine ClaDS-like and BAMM-like features. Given the general lack of support for radical shifts in diversification rates across the bird trees, it seems likely that such shifts are rare, if they occur at all. Therefore, identifying them would presumably require analyses of larger trees than the ones examined here. However, it also seems likely that the two processes interact, such that it becomes more difficult to detect major shifts when the gradual changes are not accounted for. Thus, it is possible that there are major shifts in the bird trees that our analyses failed to detect because of

shortcomings in the models. We will have to await future analyses using more sophisticated models before we know whether this is the case.

10.5 Discretizing punctuated-change models

Computing likelihoods for punctuated models of diversification by integrating out the rate priors using discrete approximations is potentially a very powerful approach²⁹. It allows for robust and computationally efficient MCMC inference, as long as a small number of rate categories yield sufficiently accurate likelihood estimates. This decidedly appears to be the case, especially if only changes in speciation rate are modeled; empirical analyses suggest that ten categories is quite sufficient for most problems²⁹.

Unfortunately, it is difficult to see how this approach can be extended to accommodate slowing (or increasing) diversification rates over times as in BAMM, because then it would be necessary to integrate out an infinite number of rate acceleration or deceleration processes with different starting points. This appears to be an important limitation from an empirical perspective, at least judging by the bird trees we analyzed. The LSBDS model does not fit many of the reconstructed trees well; this is undoubtedly linked to the substantial support for slowing diversification rates in most bird clades. If we restrict our attention to models of punctuated change, we find 8 trees with strong evidence favoring the BAMM model over the LSBDS model. There is not a single tree for which the evidence goes strongly in the other direction.

10.6 Sampling biases

Some of the results that emerge from our analyses are probably due, at least in part, to sampling biases. The lack of evidence for extinction rates above zero is an obvious case. Models without extinction (CRB, TDB, ClaDS0) almost always do better than models with extinction (Supplementary Figures 13–22). The most notable exception is the Lari (gulls), where the CRBD and TDBD models significantly outperform their non-extinction counterparts (Supplementary Figure 16). A similar but much weaker signal is seen in a few other groups: the Anatinae (Supplementary Figure 13), Charadrii (Supplementary Figure 14), Procellariidae (Supplementary Figure 18) and Psittacidae1 (Supplementary Figure 19). The outcome of the model comparison is generally consistent with posterior estimates of μ under the models that do accommodate extinction (CRBD and TDBD). That is, estimated extinction rates are usually low except for Lari, and to a lesser extent for the other groups that weakly favor models that accommodate extinction. Interestingly, Lari is also unusual in that there is evidence for accelerating speciation rates ($z > 0$). However, this occurs only in the TDB model, and is probably an artefact of not accounting for extinction, as extinction rates noticeably above zero are expected to lead to an apparent acceleration of speciation rates close to the present in reconstructed trees²⁵.

The lack of support for extinction in our analyses is consistent with results from previous diversification studies⁵². Given the overwhelming evidence for frequent extinction in the fossil record, these results are not plausible. Clearly, current phylogenetic diversification analyses tend to underestimate extinction rates. An important factor that may contribute toward such underestimation of extinction rates is the *diversified sampling* bias⁵⁴. This is the tendency of biologists to systematically select leaves of phylogenetic trees in such a way that the diversity represented in the sampled tree is maximized, instead of choosing leaves randomly as assumed by standard birth-death models. The diversified sampling bias will lead to pruning of the most recent splits from the complete reconstructed tree. The more incomplete the sampling is, the deeper the period that is devoid of splits will extend into the past. In the most extreme cases, the sampled tree will look like a bush: most of the splits will be close to the root of the tree, and all the leaves will sit on long terminal branches. If one analyzes a tree sampled to maximize diversity under a model assuming random sampling, extinction rates can be substantially underestimated⁵⁴. Failing to account for diversified sampling bias can also result in severe biases in divergence time estimates^{55,56}.

The bird trees we examined here represent fairly complete samples of extant species (from about half of the species and up), with no obvious biases (Table 6). Nevertheless, it is easy to show that they are characterized to some extent by diversified sampling. In total, the bird data include 9,993 species, 6,670 of which were sampled for sequencing⁴⁴. Only the latter were included in the trees we examined here. If the sequenced species were chosen randomly, then they would include approximately the same number of genera as any random sample of 6,670 species. However, the sequenced species cover as many as 1,880 genera, while 10,000 random samples of 6,670 species included only 1,759 genera on average (range 1,703 to 1,808). Thus, the sequenced set has significantly fewer species per genus than expected by chance, the type of bias we would predict from diversified sampling.

A similar bias that may also affect our results is that incipient species, sibling species and subspecies are likely to be underrepresented in the data. Some of those lineages will eventually develop into true species, and should be included in estimates of speciation and extinction rates at the species level. Unacknowledged diversified sampling around or just above the species level is linked to the phenomenon of *protracted speciation*⁵⁷.

Interestingly, diversified sampling bias that is not accounted for correctly could also contribute to the strong support for slowing diversification rates, even though there are also plausible biological explanations for such patterns⁵³. To understand why diversified sampling may give the impression of slowing diversification, consider that the extinction rate estimates are largely based on the apparent acceleration of speciation towards the recent seen in surviving trees because there has not been time enough for extinction of the side lineages that will eventually disappear²⁵. Diversified sampling would systematically remove the evidence for this apparent acceleration in speciation rates.

As one might expect, there is also a link between the posterior estimates of extinction and the evidence for slowing diversification rates. Specifically, models that accommodate slowing diversification rates (TDBD, ClaDS models and BAMM) are also

associated with distinctly higher estimates of initial extinction rates than models that do not (LSBDS, CRBD) (Supplementary Figures 13–22). How this link might be affected by diversified sampling biases is currently unclear. Pursuing this topic further would be outside of the scope of the current paper. However, we do note that it is relatively straightforward to modify our script to account for diversified sampling according to the model suggested by Höhna et al.⁵⁴, or potentially even more realistic models.

Some recent papers have suggested that unrealistically low extinction rate estimates may be the result of applying models that do not account for the heterogeneity across lineages in diversification rates that characterize most phylogenetic trees^{58,31}. If this were true, then one would expect extinction rate estimates in our analyses to be higher for models that accommodate lineage-specific variation than for comparable models that do not. However, this is not the case. For instance, extinction rate estimates are similar or lower for LSBDS than for the comparable CRBD model (Supplementary Figures 13–22). Similarly, the estimates are similar or lower for ClaDS1 than for CRBD. The initial extinction rate estimates of the ClaDS2 and BAMM models are best compared to the analogous estimates for the TDBD model, which does not accommodate variation across lineages in diversification rates. These estimates are similar for most bird trees (less clear for BAMM than for ClaDS2), further supporting the conclusion that unaccommodated heterogeneity in diversification rates is not responsible for the low extinction rate estimates. Of course, we cannot exclude the possibility that even more sophisticated models than the ones examined here could show that extinction rates are underestimated because of shortcomings in the modeling of across-lineage variation in diversification rates.

10.7 Statistical power

Reconstructed trees carry only a limited amount of information about absolute speciation and extinction rates. This is illustrated well by the underestimation of extinction rates discussed above. For really powerful analyses of diversification processes, we need trees that include data from the fossil record, that is, observations of both extinct and extant lineages⁵⁹. In most cases, however, observation of extinct lineages is not possible, or the information about extinct lineages is bound to be very incomplete, so we need to extract as much information as possible from trees that only (or mainly) comprise surviving lineages. There are several ways in which analyses of such trees can be improved. Addressing sampling biases appropriately would be an important step in the right direction. Making the model of the diversification process itself more realistic, for instance by combining gradual and punctuated change as suggested above, would be another. However, the most obvious way to improve the analyses would be to increase the amount of data.

A natural way of extending the present work in this direction would be to opt for a hierarchical model-averaging approach, in which all trees in a set, such as the bird clades, would be analyzed simultaneously. Specifically, each tree would randomly choose from a mixture of all available models, while the mixture proportions and the hyper-parameters tuning the priors over model-specific parameters would themselves be estimated across trees, using a hierarchical modeling design. Global estimation of hyper-priors and mixture weights based on the whole collection of trees is an efficient way to fit the priors to the true prevalence of alternative modes of diversification and the true variation in parameter values present in the data and therefore should result in well-calibrated model selection. Joint analysis of all trees would also make it possible to collect the weak signals disseminated across the many small trees of the analysis. Such developments are exactly what the probabilistic programming framework introduced here is meant to facilitate.

A completely different approach would be to analyze larger portions of the tree of life. For instance, our analyses of 40 bird clades could have been replaced with a single analysis of the entire bird tree, doubling the coverage of bird species (from 5,000 to 10,000). Such an analysis would not only include more of the variation seen across terminal clades, it would also add data on the deeper splits in the tree. These splits could potentially inform the model about long-term macroevolutionary patterns that could not be detected in analyses of only terminal clades, regardless of how sophisticated. For instance, it has been suggested that the bird radiation as a whole is characterized by rare but major boosts in diversification rates, presumably linked to key innovations opening up new ecological niches⁴⁴. If these upward jumps in diversification rates are substantial enough, it could explain why there is overwhelming support for slowing diversification rates in individual bird clades, even though the rates appear to be accelerating over the bird tree as a whole⁴⁴. Such a mega-analysis would have to be based on a model that is more sophisticated than the ones explored here. Minimally, it would have to account for both gradual and punctuated change in diversification rates. Ideally, it would also account for variation across lineages in the strength of the slowing forces on diversification, and in the rate of gradual change in speciation and extinction rates. Again, such developments are well supported by the probabilistic programming framework, although it is still an open question whether current inference strategies are efficient enough or whether further refinement is needed.

References

1. Gilks, W. R., Thomas, A. & Spiegelhalter, D. J. A language and program for complex Bayesian modelling. *The Statistician* **43**, 169–177 (1994).
2. Bishop, C. M. *Pattern Recognition and Machine Learning* (2006, New York, USA, Springer).
3. Minka, T. et al. Infer.NET 0.3 (2018). Microsoft Research Cambridge. <http://dotnet.github.io/infer>.
4. Goodman, N. D. The principles and practice of probabilistic programming. *ACM SIGPLAN Notices* **48**, 399–402 (2013).
5. Gordon, A. D., Henzinger, T. A., Nori, A. V. & Rajamani, S. K. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181 (ACM, 2014).
6. Murray, L. M., Lundén, D., Kudlicka, J., Broman, D. & Schön, T. B. Delayed sampling and automatic Rao–Blackwellization of probabilistic programs. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 21, page 10 (Lanzarote, 2018).
7. Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K. & Tenenbaum, J. B. Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, page 220–229 (AUAI Press, Arlington, Virginia, USA, 2008).
8. Pfeffer, A. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report* **137**, 96 (2009).
9. Goodman, N. D. & Stuhlmüller, A. The design and implementation of probabilistic programming languages. <http://dipp1.org> (2014). Accessed: 2020-5-12.
10. Wood, F., Meent, J. W. & Mansinghka, V. A new approach to probabilistic programming inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 1024–1032 (Reykjavik, Iceland, 2014).
11. Mansinghka, V., Selsam, D. & Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. Preprint at <https://arxiv.org/abs/1404.0099> (2014).
12. Tran, D. et al. Edward: A library for probabilistic modeling, inference, and criticism. Preprint at <https://arxiv.org/abs/1610.09787> (2016).
13. Bingham, E. et al. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research* **20**, 1–6 (2019).
14. Murray, L. M. & Schön, T. B. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control* **46**, 29–43 (2018).
15. Carpenter, B. et al. Stan: A probabilistic programming language. *Journal of Statistical Software* **76** (2017).
16. Broman, D. A Vision of Miking: Interactive programmatic modeling, sound language composition, and self-learning compilation. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, pages 55–60 (ACM, 2019).
17. van de Meent, J.-W., Paige, B., Yang, H. & Wood, F. An introduction to probabilistic programming. Preprint at <https://arxiv.org/abs/1809.10756> (2018).
18. PhyJSON—a simple JSON format for phylogenetic data. https://github.com/kudlicka/nexus2phyjson/blob/master/doc/phyjson_format_description.md (2018). Accessed: 2020-04-17.
19. BiSSE trees from MesquiteCore. <https://github.com/MesquiteProject/MesquiteCore/blob/master/Resources/examples/Diversification/06-BiSSEtrees.nex> (2009). Accessed: 2020-04-16.
20. Whale tree from BAMM. <https://github.com/macroeolution/bamm/blob/master/examples/diversification/whales/whaletree.tre> (2013). Accessed: 2020-04-16.
21. Diversitree raw content from GitHub. <https://raw.githubusercontent.com/richfitz/diversitree/master/pub/example/data/primates-10.nex> (2011). Accessed: 2020-04-16.
22. Gernhard, T. The conditioned reconstructed process. *Journal of Theoretical Biology* **253**, 769–778 (2008).
23. Semple, C. & Steel, M. *Phylogenetics* (Oxford University Press, Oxford, 2003).
24. Yule, G. U. A mathematical theory of evolution, based on the conclusions of Dr. JC Willis, FRS. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character* **213**, 21–87 (1924).

25. Nee, S. Birth-death models in macroevolution. *Annual Review of Ecology, Evolution and Systematics* **37**, 1–17 (2006).
26. Feller, W. Die Grundlagen der Volterraschen Theorie des Kampfes ums Dasein in wahrscheinlichkeitstheoretischer Behandlung. *Acta Biotheoretica* **5**, 11–40 (1939).
27. Kendall, D. G. On the generalized "birth-and-death" process. *The Annals of Mathematical Statistics* **19**, 1–15 (1948).
28. Rabosky, D. L. Automatic detection of key innovations, rate shifts, and diversity-dependence on phylogenetic trees. *PLoS ONE* **9**, e89543 (2014).
29. Höhna, S. et al. A Bayesian approach for estimating branch-specific speciation and extinction rates. Preprint at <https://biorxiv.org/content/10.1101/555805v1> (2019).
30. Maliet, O., Hartig, F. & Morlon, H. A model with many small shifts for estimating species-specific diversification rates. *Nature Ecology & Evolution* **3**, 1086–1092 (2019).
31. Morlon, H., Parsons, T. L. & Plotkin, J. B. Reconciling molecular phylogenies with the fossil record. *Proceedings of the National Academy of Sciences of the United States of America* **108**, 16327–16332 (2011).
32. Moore, B. R., Höhna, S., May, M. R., Rannala, B. & Huelsenbeck, J. P. Critically evaluating the theory and performance of Bayesian analysis of macroevolutionary mixtures. *Proceedings of the National Academy of Sciences of the United States of America* **113**, 9569–9574 (2016).
33. Laudanno, G., Haegeman, B., Rabosky, D. L. & Etienne, R. S. Detecting lineage-specific shifts in diversification: A proper likelihood approach. *Systematic Biology* **69**, 000–000 (2020).
34. Maddison, W. P., Midford, P. E. & Otto, S. P. Estimating a binary character's effect on speciation and extinction. *Systematic Biology* **56**, 701–710 (2007).
35. Yang, Z. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. *Journal of Molecular evolution* **39**, 306–314 (1994).
36. Barido-Sottani, J., Vaughan, T. G. & Stadler, T. A multitype birth–death model for Bayesian inference of lineage-specific birth and death rates. *Systematic Biology* **69**, 973–986 (2020).
37. Maliet, O. & Morlon, H. Fast and accurate estimation of species-specific diversification rates using data augmentation. Preprint at <https://www.biorxiv.org/content/10.1101/2020.11.03.365155v1> (2020).
38. Rabosky, D. L. et al. BAMMtools: an R package for the analysis of evolutionary dynamics on phylogenetic trees. *Methods in Ecology and Evolution* **5**, 701–707 (2014).
39. Lundén, D., Broman, D., Ronquist, F. & Murray, L. M. Automatic alignment of Sequential Monte Carlo inference in higher-order probabilistic programs. Preprint at <https://arxiv.org/abs/1812.07439> (2018).
40. Kudlicka, J., Murray, L. M., Ronquist, F. & Schön, T. B. Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence 2019*, volume 2019, page 11 (Tel Aviv, Israel, 2019).
41. Wigren, A., Risuleo, R. S., Murray, L. & Lindsten, F. Parameter elimination in particle Gibbs sampling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox & R. Garnett, editors, *Advances in Neural Information Processing Systems* **32**, pages 8918–8929 (Curran Associates, Inc., 2019).
42. Del Moral, P., Jasra, A., Lee, A., Yau, C. & Zhang, X. The alive particle filter and its use in particle Markov chain Monte Carlo. *Stochastic Analysis and Applications* **33**, 943–974 (2015).
43. Morlon, H. et al. RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution* **7**, 589–597 (2016).
44. Jetz, W., Thomas, G. H., Joy, J. B., Hartmann, K. & Mooers, A. O. The global diversity of birds in space and time. *Nature* **491**, 444–448 (2012).
45. Doucet, A., Pitt, M. K., Deligiannidis, G. & Kohn, R. Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *Biometrika* **102**, 295–313 (2015).
46. Kish, L. *Survey Sampling* (John Wiley & Sons, Inc., New York, NY, USA; London, UK, 2014).
47. Murray, L. M., Jones, E. M. & Parslow, J. On disturbance state-space models and the particle marginal Metropolis-Hastings sampler. *SIAM/ASA Journal on Uncertainty Quantification* **1**, 494–521 (2013). Publisher: Society for Industrial and Applied Mathematics.

48. Gelman, A. et al. *Bayesian Data Analysis* (CRC Press, Boca Raton, FL, USA, 2014), 3rd edition.
49. Lindholm, A., Zachariah, D., Stoica, P. & Schön, T. B. Data consistency approach to model validation. In *IEEE Access*, volume 7, pages 59788–59796 (2019).
50. McPeck, M. The ecological dynamics of clade diversification and community assembly. *The American Naturalist* **172**, E270–E284 (2008).
51. Weir, J. T. Divergent timing and patterns of species accumulation in lowland and highland neotropical birds. *Evolution* **60**, 842–855 (2006).
52. Pyron, R. A. & Burbrink, F. T. Phylogenetic estimates of speciation and extinction rates for testing ecological and evolutionary hypotheses. *Trends in Ecology & Evolution* **28**, 729–736 (2013).
53. Moen, D. & Morlon, H. Why does diversification slow down? *Trends in Ecology & Evolution* **29**, 190–197 (2014).
54. Höhna, S., Stadler, T., Ronquist, F. & Britton, T. Inferring speciation and extinction rates under different sampling schemes. *Molecular Biology and Evolution* **28**, 2577–2589 (2011).
55. Zhang, C., Stadler, T., Klopstein, S., Heath, T. A. & Ronquist, F. Total-evidence dating under the fossilized birth-death process. *Systematic Biology* **65**, 228–249 (2016).
56. Ronquist, F., Lartillot, N. & Phillips, M. J. Closing the gap between rocks and clocks using total-evidence dating. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* **371**, 20150136 (2016).
57. Rosindell, J., Cornell, S. J., Hubbell, S. P. & Etienne, R. S. Protracted speciation revitalizes the neutral theory of biodiversity. *Ecology Letters* **13**, 716–727 (2010).
58. Rabosky, D. L. Extinction rates should not be estimated from molecular phylogenies. *Evolution* **64**, 1816–1824 (2010).
59. Quental, T. B. & Marshall, C. R. Diversity dynamics: molecular phylogenies need the fossil record. *Trends in Ecology & Evolution* **25**, 434–441 (2010).

Paper II



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

Please cite this version:

J. Kudlicka, L. M. Murray, F. Ronquist, and T. B. Schön. Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In *Conference on Uncertainty in Artificial Intelligence*, 2019.

```
@inproceedings{kudlicka2019probabilistic,  
  title={Probabilistic programming for birth-death models of evolution using  
    an alive particle filter with delayed sampling},  
  author={Kudlicka, Jan and Murray, Lawrence M. and Ronquist, Fredrik and  
    Sch\"on, Thomas B.},  
  booktitle={Conference on Uncertainty in Artificial Intelligence},  
  year={2019}  
}
```

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.

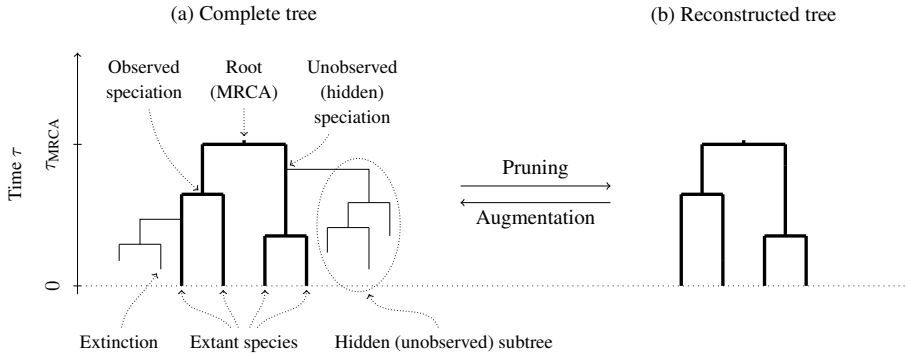


Figure 1: Complete (on the left) vs. reconstructed (on the right) tree. The reconstructed tree shows only the evolution of the extant species.

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:

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

The program may use random variables as though any ordinary variable, and control the flow of the execution as shown in the following example:

```

if  $x > 0.5$  then
   $y \sim \text{Normal}(x, 1)$ 
else
   $y \sim \text{Exponential}(1)$ 
end if

```

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:

```

 $x \sim \text{Normal}(0, 1)$ 
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., $v_{\sigma[1]}, \dots, v_{\sigma[|\sigma|]}$. In a similar manner, we will use $\{V_i = v_i\}_{i \in \sigma}$ to denote $V_{\sigma[1]} = v_{\sigma[1]}, \dots, V_{\sigma[|\sigma|]} = v_{\sigma[|\sigma|]}$.

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.

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

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

where $\kappa = \text{Ne}(\{v_i\}_{i \in \sigma})$ 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 $p(\{v_i\}_{i \notin \gamma} | V_{\gamma[1]} = y_1, \dots, V_{\gamma[T]} = y_T)$, 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, $\text{Cat}()$ 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.

Algorithm 1 Bootstrap particle filter (BPF).

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

function PROPAGATE( $v$ ) ▷ Run until next observe
   $\kappa \leftarrow \text{Ne}(v)$ 
  while  $\kappa \notin \gamma$  do
     $v[\kappa] \sim p_{\kappa}(\cdot | \text{Pa}(v))$ 
     $\kappa \leftarrow \text{Ne}(v)$ 
  end while
  return  $v$ 
end function
```

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

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

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,

Algorithm 2 Alive particle filter (APF).

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

but propagations made using this particle are included in P_t , and used to calculate the unbiased estimate of the marginal likelihood $p(y_{1:T})$:

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

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

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

Phylogenetic 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.

Phylogenetic trees are unordered (i.e. there is no specific ordering of the children of each internal node) and usually include the labels for the extant species. To derive the likelihood of a complete labeled phylogenetic tree \mathcal{T} , let us first assume that the tree is ordered and unlabeled. Let \mathcal{T}_r denote the subtree rooted at the node r , and $\text{Ch}(r)$ 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):

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

where Δ_r is the length of the branch between the node r and its parent. 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, unordered and labeled phylogeny \mathcal{T} is given by

$$p(\mathcal{T}) = \frac{2^{S+1}}{C!} p(\mathcal{T}_{\text{root}}),$$

where S is the number of speciation events (excluding the root) and C is the number of extant species. The factor 2^{S+1} represents the number of all possible orderings of the tree and there are $C!$ permutations of the labels of the extant species. The likelihood can be conveniently written as

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

where we have introduced L to denote the sum of all branch lengths and X to denote the number of extinction events. The likelihood in other birth-death models that admit varying rates and/or include the state can be derived in a similar way.

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. 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 there are two possible orderings of the children created by a hidden speciation event on an observed branch.

If the examined branch ends with a speciation event, the algorithm observes $0 \sim \text{Exponential}(\lambda)$. 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}' that is compatible with the observation (i.e. without any extant species in the hidden subtrees) is given by

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

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}' is given by

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

where S_{obs} is the number of observed speciation events. The factor $2^{S_{\text{obs}}+1}/C!$, related to the number of possible orderings and the number of labeling permutations, is used as the initial weight of each proposal. The factor $2^{H'}$ corresponds to doubling the weight for each hidden subtree, the factor $\lambda^{S_{\text{obs}}}$ is due to observing $0 \sim$

Exponential(λ) 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

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

The factor $2^{S_{\text{obs}}+1}/C!$ in $w(\mathcal{T}')$ is constant for all proposals and we will omit it from the weight in the algorithms and experiments.

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.

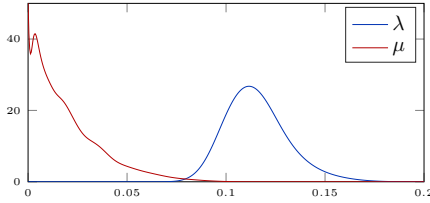


Figure 2: The posterior distributions for the speciation and extinction rates for the cetaceans using the CRBD model.

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 $\{\hat{Z}_m\}_{m=1}^M$ of the marginal likelihood and calculated the relative effective sample size (RESS):

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

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 $\{\hat{Z}_m\}_m$. We also calculated the sample variance $\text{var} \log \hat{Z}$.

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 5 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.

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 $k_\lambda^{(m)}, \theta_\lambda^{(m)}, k_\mu^{(m)}$

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** **continue** **end if** $c_{\text{hs}} \sim \text{Poisson}(\lambda \Delta_r)$ **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** **end if**

double the weight

end for **if** r has children **then** **observe** $0 \sim \text{Exponential}(\lambda)$ **end if** **observe** $0 \sim \text{Poisson}(\mu \Delta_r)$ **end for****function** BRANCHSURVIVES(τ, λ, μ) $\Delta \sim \text{Exponential}(\mu)$ **if** $\Delta \geq \tau$ **then** **return true** **end if** $c_b \sim \text{Poisson}(\lambda \Delta)$ **for** $i \leftarrow 1$ **to** c_b **do** $\tau' \sim \text{Uniform}(\tau - \Delta, \tau)$ **if** BRANCHSURVIVES(τ', λ, μ) **then** **return true** **end if** **end for** **return false****end function**

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:

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

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 Gamma(1, 1) as the prior for $\lambda_0, \lambda_1, \mu_0$ and μ_1 , and 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 $\text{var} \log \hat{Z}$.

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 $\text{var} \log \hat{Z}$ 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.

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.

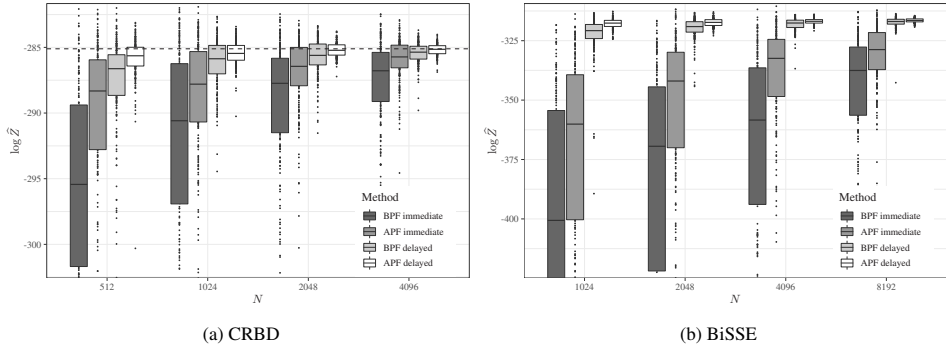


Figure 3: Box plot of $\log \hat{Z}$ for the CRBD (on the left) and BiSSE (on the right) model for different number of particles (N) and methods. The lower and upper hinges correspond to the first and third quartile, whereas the midhinge corresponds to the median. Values outside of the interquartile 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 $M = 200$.

N	Bootstrap particle filter (BPF)						Alive particle filter (APF)							
	Immediate sampling			Delayed sampling			Immediate sampling				Delayed sampling			
	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 $M = 200$.

N	Bootstrap particle filter (BPF)						Alive particle filter (APF)							
	Immediate sampling			Delayed sampling			Immediate sampling				Delayed sampling			
	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

- 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.

- M. K. Pitt, R. dos Santos Silva, P. Giordani, and R. Kohn. On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. *Journal of Econometrics*, 171(2):134–151, 2012.
- 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.

SUPPLEMENTARY MATERIAL

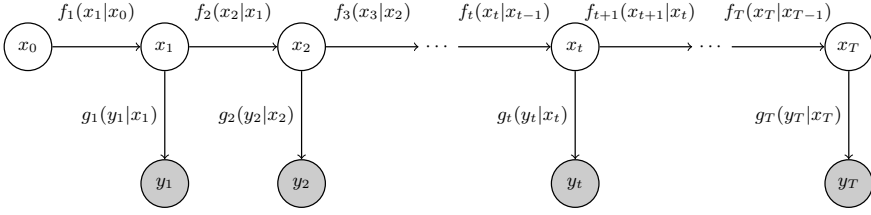


Figure 4: Graphical model of the state space model.

A Proof of the unbiasedness of the marginal likelihood estimator of the extended APF

In this section we prove that the marginal likelihood estimator

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

produced by the extended alive particle filter (APF) for the state space model (Figure 4)

$$\begin{aligned} x_0 &\sim p(x_0), \\ x_t &\sim f_t(x_t|x_{t-1}), \text{ for } t = 1, 2, \dots, T, \\ y_t &\sim g_t(y_t|x_t), \end{aligned}$$

is unbiased in the sense that $\mathbb{E}[\widehat{Z}] = p(y_{1:T})$.

The structure of our proof is similar to that of Pitt et al. (2012) for the Auxiliary Particle Filter. Let $\mathcal{F}_t = \{x_t^{(n)}, w_t^{(n)}\}_{n=1}^N$ denote the internal state of the particle filter, i.e., the states and weights of all particles, at time t .

Lemma 1.

$$\mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1} \middle| \mathcal{F}_{t-1} \right] = \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_t | x_{t-1}^{(n)}).$$

Proof. In the interest of brevity, we will omit conditioning on \mathcal{F}_{t-1} in the notation. For each particle, the APF constructs a candidate sample x' by drawing a sample from $\{x_{t-1}^{(n)}\}$ with the probabilities proportional to the weights $\{w_{t-1}^{(n)}\}$ and propagating it forward to time t such that

$$x' \sim \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x' | x_{t-1}^{(n)}).$$

If $g_t(y_t|x') = 0$, the candidate sample is rejected and the procedure is repeated until acceptance (when $g_t(y_t|x') > 0$).

Let $A_t = \{x' : g_t(y_t|x') > 0\}$. The acceptance probability p_{A_t} is then given by

$$p_{A_t} = \int \mathbf{1}_{A_t}(x') \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x' | x_{t-1}^{(n)}) dx',$$

where $\mathbf{1}$ denotes the indicator function.

The accepted samples are distributed according to the following distribution:

$$x_t \sim \frac{\mathbf{1}_{A_t}(x_t)}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x_t | x_{t-1}^{(n)}).$$

The expected value of the weight $w_t = g_t(y_t|x_t)$ of an accepted sample is given by

$$\mathbb{E}[w_t] = \int g_t(y_t|x_t) \frac{\mathbf{1}_{A_t}(x_t)}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x_t|x_{t-1}^{(n)}) dx_t.$$

The factor $\mathbf{1}_{A_t}(x_t)$ can be omitted since $\mathbf{1}_{A_t}(x_t) = 0 \Leftrightarrow g_t(y_t|x_t) = 0$, resulting in

$$\begin{aligned} \mathbb{E}[w_t] &= \int \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x_t|x_{t-1}^{(n)}) g_t(y_t|x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} \int f_t(x_t|x_{t-1}^{(n)}) g_t(y_t|x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_t|x_{t-1}^{(n)}). \end{aligned}$$

The APF repeats drawing new samples until $N + 1$ samples have been accepted. The total number of draws of candidate samples at time t , P_t , is itself a random variable distributed according to the negative binomial distribution

$$P(P_t = D) = \binom{D-1}{(N+1)-1} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)}$$

with the support $D \in \{N+1, N+2, N+3, \dots\}$.

Finally, using the fact that $\mathbb{E}[w_t]$ does not depend on the value of P_t ,

$$\begin{aligned} \mathbb{E}\left[\frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1}\right] &= \sum_{D=N+1}^{\infty} \frac{N\mathbb{E}[w_t]}{D-1} P(P_t = D) = \sum_{D=N+1}^{\infty} \frac{N\mathbb{E}[w_t]}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N\mathbb{E}[w_t] \sum_{D=N+1}^{\infty} \frac{1}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N\mathbb{E}[w_t] \sum_{D=N+1}^{\infty} \frac{1}{D-1} \frac{(D-1)(D-2)!}{N(N-1)!(D-(N+1))!} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= \mathbb{E}[w_t] p_{A_t}^{N+1} \sum_{D=N+1}^{\infty} \binom{D-2}{D-(N+1)} (1-p_{A_t})^{D-(N+1)} \\ &\text{(using the binomial theorem)} \\ &= \mathbb{E}[w_t] p_{A_t}^{N+1} p_{A_t}^{-N} = \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_t|x_{t-1}^{(n)}) p_{A_t} \\ &= \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_t|x_{t-1}^{(n)}). \end{aligned}$$

□

Lemma 2.

$$\mathbb{E}\left[\frac{\sum_{n=1}^N w_t^{(n)} p(y_{t+1:t'}|x_t^{(n)})}{P_t - 1} \middle| \mathcal{F}_{t-1}\right] = \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'}|x_{t-1}^{(n)}).$$

Proof. Similar to the proof of Lemma 1 we have that

$$\begin{aligned} \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] &= \int \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x_t | x_{t-1}^{(n)}) g_t(y_t | x_t) p(y_{t+1:t'} | x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} \int f_t(x_t | x_{t-1}^{(n)}) g_t(y_t | x_t) p(y_{t+1:t'} | x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}) \end{aligned}$$

and using this result we have that

$$\begin{aligned} \mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)} p(y_{t+1:t'} | x_t^{(n)})}{P_t - 1} \right] &= \sum_{D=N+1}^{\infty} \frac{N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)]}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] \sum_{D=N+1}^{\infty} \frac{1}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] \frac{p_{A_t}}{N} = N \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}) \frac{p_{A_t}}{N} \\ &= \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}). \end{aligned}$$

□

Lemma 3.

$$\mathbb{E} \left[\prod_{t'=t-h}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{F}_{t-h-1} \right] = \sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{\sum_{m=1}^N w_{t-h-1}^{(m)}} p(y_{t-h:t} | x_{t-h-1}^{(n)}).$$

Proof. By induction.

The base step for $h = 0$ was proved in Lemma 1.

In the induction step, let us assume that the equality holds for h and prove it for $h + 1$:

$$\begin{aligned} \mathbb{E} \left[\prod_{t'=t-h-1}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{F}_{t-h-2} \right] &= \mathbb{E} \left[\mathbb{E} \left[\prod_{t'=t-h}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{F}_{t-h-1} \right] \frac{\sum_{n=1}^N w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} \middle| \mathcal{F}_{t-h-2} \right] \\ &\quad \text{(using the induction assumption)} \\ &= \mathbb{E} \left[\sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{\sum_{m=1}^N w_{t-h-1}^{(m)}} p(y_{t-h:t} | x_{t-h-1}^{(n)}) \frac{\sum_{n=1}^N w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} \middle| \mathcal{F}_{t-h-2} \right] \\ &= \mathbb{E} \left[\sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} p(y_{t-h:t} | x_{t-h-1}^{(n)}) \middle| \mathcal{F}_{t-h-2} \right] \\ &\quad \text{(using Lemma 2)} \\ &= \sum_{n=1}^N \frac{w_{t-h-2}^{(n)}}{\sum_{m=1}^N w_{t-h-2}^{(m)}} p(y_{t-h-1:t} | x_{t-h-2}^{(n)}). \end{aligned}$$

□

Theorem 1.

$$\mathbb{E} \left[\prod_{t=1}^T \frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1} \right] = p(y_{1:T}).$$

Proof. Using Lemma 3 with $t = T, h = T - 1$ and

$$\mathbb{E} \left[\frac{1}{N} \sum_{n=1}^N p \left(y_{1:T} \mid x_0^{(n)} \right) \right] = p(y_{1:T}).$$

□

B Generative model for CRBD

The pseudocode for generating phylogenetic trees using the CRBD model is listed in Algorithm 4.

Algorithm 4 Pseudocode for generating trees using the CRBD model.

```

function CRBD( $\tau_{\text{orig}}$ )
  return ( $\tau_{\text{orig}}, \{\text{CRBD}'(\tau_{\text{orig}})\}$ )
end function

function CRBD'( $\tau$ )
   $\Delta \sim \text{Exponential}(\lambda + \mu)$ 
   $\tau' \leftarrow \tau - \Delta$ 
  if  $\tau' < 0$  then
    return (0,  $\emptyset$ )
  end if
   $e \sim \text{Cat} \left( p_1 = \frac{\lambda}{\lambda + \mu}, p_2 = \frac{\mu}{\lambda + \mu} \right)$ 
  if  $e = 1$  then
    return ( $\tau', \{\text{CRBD}'(\tau'), \text{CRBD}'(\tau')\}$ )
  else
    return ( $\tau', \emptyset$ )
  end if
end function

```

C Relevant conjugacy relationships

C.1 Negative binomial and Lomax distribution

Negative binomial distribution

Parameters: number of successes $k > 0$ before the experiment is stopped, probability of success $p \in (0, 1)$

Probability mass function:

$$f(r|k, p) = \binom{r+k-1}{k-1} p^k (1-p)^r \text{ for } r \in \mathbb{N} \cup \{0\},$$

where r is the number of failures.

Lomax distribution

Parameters: scale $\lambda > 0$, shape $\alpha > 0$

Probability density function:

$$f(\Delta|\lambda, \alpha) = \frac{\alpha}{\lambda} \left(1 + \frac{\Delta}{\lambda} \right)^{-(\alpha+1)} \text{ for } \Delta \geq 0$$

C.2 Conjugacy relationships

Gamma-Poisson mixture

Prior distribution: $\nu \sim \text{Gamma}(k, \theta)$ with the probability density function

$$f(\nu|k, \theta) = \frac{1}{\Gamma(k)\theta^k} \nu^{k-1} e^{-\nu/\theta} \text{ for } \nu > 0$$

Likelihood: $n \sim \text{Poisson}(\nu\Delta)$ with the probability mass function

$$f(n|\lambda) = \frac{\lambda^n}{n!} e^{-\lambda} \text{ for } n \in \mathbb{N} \cup \{0\},$$

where $\lambda = \nu\Delta$.

Prior predictive distribution ($k \in \mathbb{N}$):

$$\begin{aligned} f(n|k, \theta) &= \int_0^\infty \frac{1}{\Gamma(k)\theta^k} \nu^{k-1} e^{-\nu/\theta} \times \frac{(\nu\Delta)^n}{n!} e^{-\nu\Delta} d\nu = \frac{\Delta^n}{n!(k-1)!\theta^k} \int_0^\infty \nu^{n+k-1} e^{-\nu(1/\theta+\Delta)} d\nu \\ &= \frac{\Delta^n}{n!(k-1)!\theta^k} \left(\frac{1}{\theta} + \Delta\right)^{-(n+k)} (n+k-1)! = \binom{n+k-1}{k-1} \left(\frac{1}{1+\Delta\theta}\right)^k \left(1 - \frac{1}{1+\Delta\theta}\right)^n \\ n|k, \theta &\sim \text{NegativeBinomial}\left(k, \frac{1}{1+\Delta\theta}\right) \end{aligned}$$

Posterior distribution:

$$\begin{aligned} f(\nu|n) &\propto \frac{1}{\Gamma(k)\theta^k} \nu^{k-1} e^{-\nu/\theta} \times \frac{(\nu\Delta)^n}{n!} e^{-\nu\Delta} \propto \nu^{k+n-1} e^{-\nu(1/\theta+\Delta)} = \nu^{(k+n)-1} e^{-\nu/(\frac{\theta}{1+\Delta\theta})} \\ \nu|n &\sim \text{Gamma}\left(k+n, \frac{\theta}{1+\Delta\theta}\right) \end{aligned}$$

Gamma-exponential mixture

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

Likelihood: $\Delta \sim \text{Exponential}(\nu)$ with the probability density function

$$f(\Delta|\nu) = \nu e^{-\nu\Delta} \text{ for } \Delta \geq 0$$

Prior predictive distribution:

$$\begin{aligned} f(\Delta|k, \theta) &= \int_0^\infty \frac{1}{\Gamma(k)\theta^k} \nu^{k-1} e^{-\nu/\theta} \times \nu e^{-\nu\Delta} d\nu = \frac{1}{\Gamma(k)\theta^k} \int_0^\infty \nu^k e^{-\nu(1/\theta+\Delta)} d\nu \\ &= \frac{1}{\Gamma(k)\theta^k} \left(\frac{1}{\theta} + \Delta\right)^{-(k+1)} \Gamma(k+1) = \frac{k}{\theta^k} \left(\frac{1}{\theta} + \Delta\right)^{-(k+1)} = k\theta(1+\Delta\theta)^{-(k+1)} \\ \Delta|k, \theta &\sim \text{Lomax}\left(\frac{1}{\theta}, k\right) \end{aligned}$$

Posterior distribution:

$$\begin{aligned} f(\nu|\Delta) &\propto \frac{1}{\Gamma(k)\theta^k} \nu^{k-1} e^{-\nu/\theta} \times \nu e^{-\nu\Delta} \propto \nu^k e^{-\nu(1/\theta+\Delta)} = \nu^{(k+1)-1} e^{-\nu/(\frac{\theta}{1+\Delta\theta})} \\ \nu|\Delta &\sim \text{Gamma}\left(k+1, \frac{\theta}{1+\Delta\theta}\right) \end{aligned}$$

D Source code

Birch is available at

<https://birch-lang.org/>

The source code for the CRBD and BiSSE models is available at

<https://github.com/kudlicka/paper-2019-probabilistic>

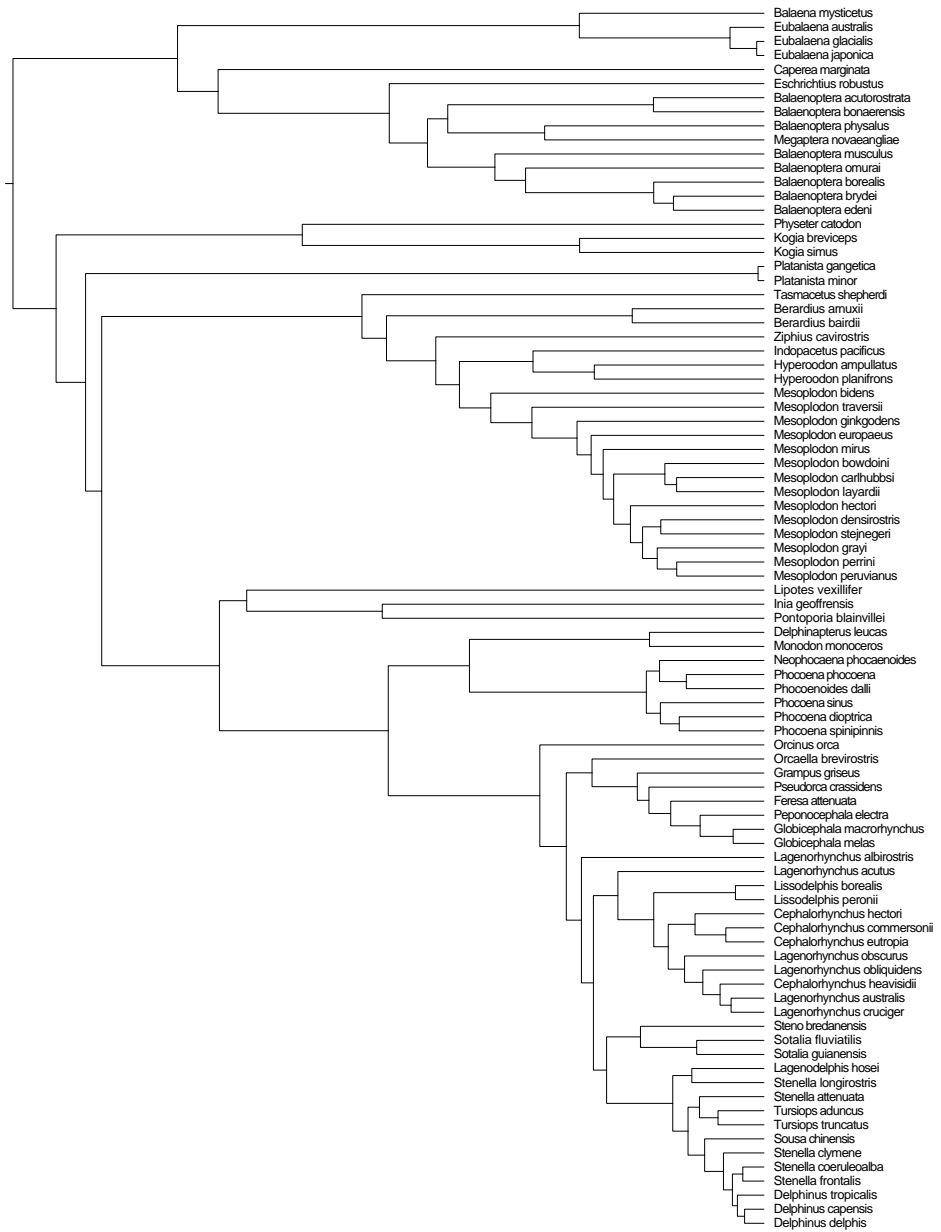


Figure 5: Phylogeny of cetaceans (whales, dolphins and porpoises).

Paper III



Delayed Sampling and Automatic Rao–Blackwellization of Probabilistic Programs

Lawrence M. Murray
Uppsala University

Daniel Lundén
KTH Royal Institute of Technology

Jan Kudlicka
Uppsala University

David Broman
KTH Royal Institute of Technology

Thomas B. Schön
Uppsala University

Abstract

We introduce a dynamic mechanism for the solution of analytically-tractable substructure in probabilistic programs, using conjugate priors and affine transformations to reduce variance in Monte Carlo estimators. For inference with Sequential Monte Carlo, this automatically yields improvements such as locally-optimal proposals and Rao–Blackwellization. The mechanism maintains a directed graph alongside the running program that evolves dynamically as operations are triggered upon it. Nodes of the graph represent random variables, edges the analytically-tractable relationships between them. Random variables remain in the graph for as long as possible, to be sampled only when they are used by the program in a way that cannot be resolved analytically. In the meantime, they are conditioned on as many observations as possible. We demonstrate the mechanism with a few pedagogical examples, as well as a linear-nonlinear state-space model with simulated data, and an epidemiological model with real data of a dengue outbreak in Micronesia. In all cases one or more variables are automatically marginalized out to significantly reduce variance in estimates of the marginal likelihood, in the final case facilitating a random-weight or pseudo-marginal-type importance sampler for parameter estimation. We have implemented the approach in Anglican and a new probabilistic programming language called Birch.

1 INTRODUCTION

Probabilistic programs extend graphical models with support for stochastic branches, in the form of conditionals, loops, and recursion. Because they are highly expressive, they pose a challenge in the design of appropriate inference algorithms. This work focuses on Sequential Monte Carlo (SMC) inference algorithms [4], extending an arc of research that includes probabilistic programming languages (PPLs) such as Venture [14], Anglican [26], Probabilistic C [18], WebPPL [8], Figaro [20], and Turing [7], as well as similarly-motivated software such as LibBi [16] and BiiPS [25].

The simplest SMC method, the bootstrap particle filter [9], requires only simulation—not pointwise evaluation—of the prior distribution. While widely applicable, it may be suboptimal with respect to Monte Carlo variance in situations where, in fact, pointwise evaluation is possible, so that other options are viable. One way of reducing Monte Carlo variance is to exploit analytical relationships between random variables, such as conjugate priors and affine transformations. Within SMC, this translates to improvements such as the locally-optimal proposal, variable elimination, and Rao–Blackwellization (see [5] for an overview). The present work seeks to automate such improvements for the user of a PPL.

Typically, a probabilistic program must be run in order to discover the relationships between random variables. Because of stochastic branches, different runs may discover different relationships, or even different random variables. While an equivalent graphical model might be constructed for any single run, it would constitute only partial observation. It may take many runs to observe the full model, if this is possible in finite time at all. We therefore seek a runtime mechanism for the solution of analytically-tractable substructure, rather than a compile-time mechanism of static analysis.

A general-purpose programming language can be augmented with some additional constructs, called *checkpoints*, to produce a PPL (see e.g. [26]). Two checkpoints are usual, denoted *sample* and *observe*. The first suggests that a value for a random variable needs to be sampled, the second that a value for a random variable is given and needs to be conditioned upon. At these checkpoints, random behavior may occur in the otherwise-deterministic execution of the program, and intervention may be required by an inference algorithm to produce a correct result.

The simplest inference algorithm instantiates a random variable when first encountered at a sample checkpoint, and updates a weight with the likelihood of a given value at an observe checkpoint. This produces samples from the prior distribution, weighted by their likelihood under the observations. It corresponds to importance sampling with the posterior as the target and the prior as the proposal. A more sophisticated inference algorithm runs multiple instances of the program simultaneously, pausing after each observe checkpoint to resample amongst executions. This corresponds to the bootstrap particle filter (see e.g. [26]).

These are *forward* methods, in the sense that checkpoints are executed in the order encountered, and sampling is myopic of future observations. The present work introduces a mechanism to change the order in which checkpoints are executed so that sampling can be informed by future observations, exploiting analytical relationships between random variables. This facilitates more sophisticated *forward-backward* methods, in the sense that information from future observations can be propagated backward through the program.

We refer to this new mechanism as *delayed sampling*. When a sample checkpoint is reached, its execution is delayed. Instead, a new node representing the random variable is inserted into a graph that is maintained alongside the running program. This graph resembles a directed graphical model of those random variables encountered so far that are involved in analytically-tractable relationships. Each node of the graph is marginalized and conditioned by analytical means for as long as possible until, eventually, it must be instantiated for the program to continue execution. This occurs when the random variable is passed as an argument to a function for which no analytical overload is provided. It is at this last possible moment that sampling is executed and the random variable instantiated.

Operations on the graph are forward-backward. The forward pass is a filter, marginalizing each latent variable over its parents and conditioning on observations, in all cases analytically. The backward pass produces a joint sample. This has some similarity to belief prop-

agation [19], but the backward passes differ: belief propagation typically obtains the marginal posterior distribution of each variable, not a joint sample. Furthermore, in delayed sampling the graph evolves dynamically as the program executes, and at any time represents only a fraction of the full model. This means that some heuristic decisions must be made without complete knowledge of the model structure.

For SMC, delayed sampling yields locally-optimal proposals, variable elimination, and Rao–Blackwellization, with some limitations, to be detailed later. At worst, it provides no benefit. There is little intrusion of the inference algorithm into modeling code, and possibly no intrusion with appropriate language support. This is important, as we consider the user experience and ergonomics of a PPL to be of primary importance.

Related work has considered analytical solutions to probabilistic programs. Where a full analytical solution is possible, it can be achieved via symbolic manipulations in Hakaru [23]. Where not, partial solutions using compile-time program transformations are considered in [17] to improve the acceptance rate of Metropolis–Hastings algorithms. This compile-time approach requires careful treatment of stochastic branches, and even then it may not be possible to propagate analytical solutions through them. Delayed sampling instead operates dynamically, at runtime. It handles stochastic branches without problems, but may introduce some additional execution overhead.

The paper is organized as follows. Section 2 introduces the delayed sampling mechanism. Section 3 provides a set of pedagogical examples and two empirical case studies. Section 4 discusses some limitations and future work. Supplementary material includes further details of the case studies and implementations.

2 METHODS

As a probabilistic program runs, its memory state evolves dynamically and stochastically over time, and can be considered a stochastic process. Let $t = 1, 2, \dots$ index a sequence of checkpoints. These checkpoints may differ across program runs (this is one of the challenges of inference for probabilistic programs, see e.g. [27]). In contrast to the two-checkpoint sample-observe formulation, we define three checkpoint types:

- `assume($X, p(\cdot)$)` to initialize a random variable X with prior distribution $p(\cdot)$,
- `observe($x, p(\cdot)$)` to condition on a random variable X with likelihood $p(\cdot)$ having some value x ,
- `value(X)` to realize a value for a random variable X previously encountered at an `assume` checkpoint.

We use the statistics convention that an uppercase character (e.g. X) denotes a random variable, while the corresponding lowercase character (e.g. x) denotes an instantiation of it.

An assume checkpoint does not result in a random variable being sampled: its sampling is delayed until later. A value checkpoint occurs the first time that a random variable, previously encountered by an assume, is used in such a way that its value is required. At this point it cannot be delayed any longer, and is sampled.

Denote the state of the running program at checkpoint t by $X_t \in \mathbb{X}_t$. This can be interpreted as the current memory state of the program. Randomness is exogenous and represented by the random process $U_t \in \mathbb{U}_t$. This may be, for example, random entropy, a pseudorandom number sequence, or uniformly distributed quasirandom numbers.

The program is a sequence of functions f_t that each maps a starting state $X_{t-1} = x_{t-1}$ and random input $U_t = u_t$ to an end state $X_t = x_t$, so that $x_t = f_t(x_{t-1}, u_t)$. Note that f_t is a deterministic function given its arguments. It is not permitted that f_t has any intrinsic randomness, only the extrinsic randomness provided by U_t .

The target distribution over X_t is $\pi_t(dx_t)$, typically a Bayesian posterior. In general, the program cannot sample from this directly. Instead, it samples x_t from some proposal distribution $q_t(dx_t)$, which in many cases is just the prior distribution $p_t(dx_t)$. Then, assuming that both π_t and q_t admit densities, it computes an associated importance weight $w_t \propto \pi_t(x_t)/q_t(x_t)$. Assuming U_t is distributed according to $\xi_t(du_t)$, we have

$$q_t(dx_t) = \int_{\mathbb{X}_{t-1}} \int_{\mathbb{U}_t} \delta_{f_t(x_{t-1}, u_t)}(dx_t) \xi_t(du_t) q_{t-1}(dx_{t-1}),$$

where δ is the Dirac measure. For brevity, we omit the subscript t henceforth, and simply update the state for the next time, as though it is mutable.

2.1 Motivation

We are motivated by variance reduction in Monte Carlo estimators. Consider some functional $\varphi(X)$ of interest. We wish to compute expectations of the form:

$$\mathbb{E}_\pi[\varphi(X)] = \int_{\mathbb{X}} \varphi(x) \pi(dx) = \int_{\mathbb{X}} \varphi(x) \frac{\pi(x)}{q(x)} q(dx).$$

Self-normalized importance sampling estimates can be formed by running the program N times and computing (where superscript n indicates the n th program run):

$$\hat{\varphi} := \sum_{n=1}^N \bar{w}^n \varphi(x^n), \quad \bar{w}^n = w^n / \sum_{n=1}^N w^n.$$

A classic aim is to reduce mean squared error:

$$\text{MSE}(\hat{\varphi}) = \mathbb{E}_q \left[(\hat{\varphi} - \mathbb{E}_\pi[\varphi(X)])^2 \right].$$

One technique to do so is *Rao–Blackwellization* (see e.g. [21, §4.2]). Assume that, amongst the state X , there is some variable X_v which has been observed to have value x_v , some set of variables X_M which can be marginalized out analytically, and some other set of variables X_R which have been instantiated previously. The functional of interest is the incremental likelihood of x_v . An estimator would usually require instantiation of $X_M^n \sim p(dx_M^n | x_R^n)$ for $n = 1, \dots, N$, and computation of:

$$\hat{Z} := \sum_{n=1}^N \bar{w}^n p(x_v | x_M^n, x_R^n).$$

The Rao–Blackwellized estimator does not instantiate X_M , but rather marginalizes it out:

$$\hat{Z}_{RB} := \sum_{n=1}^N \bar{w}^n \int p(x_v | x_M^n, x_R^n) p(dx_M^n | x_R^n).$$

By the law of total variance, $\text{var}(\hat{Z}_{RB}) \leq \text{var}(\hat{Z})$, and as \hat{Z} and \hat{Z}_{RB} are unbiased [3], $\text{MSE}(\hat{Z}_{RB}) \leq \text{MSE}(\hat{Z})$.

This form of Rao–Blackwellization is local to each checkpoint. While X_M is marginalized out, it may require instantiation at future checkpoints, and so it must also be possible to simulate $p(dx_M | x_v, x_R)$.

2.2 Delayed sampling

Delayed sampling uses analytical relationships to reorder the execution of checkpoints and reduce variance. Each observe is executed as early as possible, and the sampling associated with assume is delayed for as long as possible, to be informed by observations in between.

Alongside the state X , we maintain a graph $G = (V, E)$. This is a directed graph consisting of a set of nodes V and set of edges $E \subset V \times V$, where $(u, v) \in E$ indicates a directed edge from a parent node u to a child node v . For $v \in V$, let $\text{Pa}(v) = \{u \in V \mid (u, v) \in E\}$ denote its set of parents, and $\text{Ch}(v) = \{u \in V \mid (v, u) \in E\}$ its set of children. Associated with each $v \in V$ is a random variable X_v (part of the state, X) and prior probability distribution $p_v(dx_v | x_{\text{Pa}(v)})$, now using the subscript of X to select that part of the state associated with a single node, or set of nodes. We partition V into three disjoint sets according to three states. Let

- $I \subseteq V$ be the set of nodes in an *initialized* state,
- $M \subseteq V$ be the set of nodes in a *marginalized* state,
- $R \subseteq V$ be the set of nodes in a *realized* state.

At some checkpoint, the program would usually have instantiated all variables in V with a simulated or observed value, whereas under delayed sampling only those in R are instantiated, while those in $I \cup M$ are delayed.

We will restrict the graph G to be a forest of zero or more disjoint trees, such that each node has at most one parent. This condition is easily ensured by construction: the implementation makes anything else impossible, i.e. only relationships between pairs of random variables are coded. There are some interesting relationships that cannot be represented as trees, such as a normal distribution with conjugate prior over both mean and variance, or multivariate normal distributions. We deal with these as special cases, collecting multiple nodes into single supernodes and implementing relationships between pairs of supernodes, much like the structure achieved by the junction tree algorithm [11].

The following invariants are preserved at all times:

1. If a node is in M then its parent is in M . (1)
2. A node has at most one child in M . (2)

These imply that the nodes of M form marginalized paths: one in each of the disjoint trees of G , from the root node to a node (possibly itself) in the same tree. We will refer to the unique such path in each tree as its M -path. The node at the start of the M -path is a root node, while the node at the end is referred to as a *terminal* node. Terminal nodes have a special place in the algorithms below, and are denoted by the set T .

By the invariants, each $v \in M \setminus T$ has a child $u \in M$; let $\text{Fo}(v)$ denote the entire subtree with this child u as its root (the *forward* set). Otherwise let $\text{Fo}(v)$ be the empty set. The graph G then encodes the distribution

$$\left(\prod_{v \in I} q_v(dx_v \mid x_{\text{Pa}(v)}) \right) \left(\prod_{v \in M \setminus T} q_v(dx_v \mid x_{R \setminus \text{Fo}(v)}) \right) \times \left(\prod_{v \in T} q_v(dx_v \mid x_R) \right), \quad (3)$$

where q_v equals the prior for nodes in I , some updated distribution for nodes in M , and all nodes in R are instantiated. The distribution suggests why terminals (in the set T) are important: they are the nodes informed by all instantiated random variables up to the current point in the program, and can be immediately instantiated themselves. Other nodes in M await information to be propagated backward from their forward set before they, too, can be instantiated.

When the program reaches a checkpoint, it triggers operations on the graph (details follow):

- For $\text{assume}(X_v, p(\cdot))$, call $\text{INITIALIZE}(v, p(\cdot))$, which inserts a new node v into the graph.
- For $\text{observe}(x_v, p(\cdot))$, call $\text{INITIALIZE}(v, p(\cdot))$, then $\text{GRAFT}(v)$, which turns v into a terminal node, then $\text{OBSERVE}(v)$, which assigns the observed value to v and updates its parent by conditioning.
- For $\text{value}(X_v)$, call $\text{GRAFT}(v)$, then $\text{SAMPLE}(v)$, which samples a value for v .

Figure 1 provides pseudocode for all operations; Figure 2 illustrates their combination. Operations are of two types: *local* and *recursive*. Local operations modify a single node and possibly its parent:

- $\text{INITIALIZE}(v, p(\cdot))$ inserts a new node v into the graph. If v requires a parent, u (implied by $p(\cdot)$ having a conditional form, i.e. $p(dx_v \mid x_u)$ not $p(dx_v)$), then v is put in I and the edge (u, v) inserted. Otherwise, it is a root node and is put in M , with no edges inserted.
- $\text{MARGINALIZE}(v)$, where v is the child of a terminal node, moves v from I to M and updates its distribution by marginalizing over its parent.
- $\text{SAMPLE}(v)$ or $\text{OBSERVE}(v)$, where v is a terminal node, assigns a value to the associated random variable by either sampling or observing, moves v from M to R , and updates the distribution of its parent node by conditioning. Both $\text{SAMPLE}(v)$ and $\text{OBSERVE}(v)$ use an auxiliary function $\text{REALIZE}(v)$ for their common operations.

As shown in the pseudocode, these local operations have strict preconditions that limit their use to only a subset of the nodes of the graph, e.g. only terminal nodes may be sampled or observed. As long as these preconditions are satisfied, the invariants (1) and (2) are maintained, and the graph G encodes the representation (3). This is straightforward to check.

The recursive operations realign the M -path to establish the preconditions for any given node, so that local operations may be applied to it. These have side effects, in that other nodes may be modified to achieve the realignment. The key recursive operation is GRAFT , which combines local operations to extend the M -path to a given node, making it a terminal node. Internally, GRAFT may call another recursive operation, PRUNE , to shorten the existing M -path by realizing one or more variables.

3 EXAMPLES

We have implemented delayed sampling in Anglican (see also [13]) and a new PPL called Birch. Details are

Program	Checkpoint	Local operations	Commentary
<pre>x ~ N(0,1); y ~ N(x,1); z ~ N(y,1);</pre>	<p>assume(X)</p> <p>assume(Y)</p> <p>observe(z)</p>	<p>INITIALIZE(X)</p> <p>INITIALIZE(Y)</p> <p>INITIALIZE(Z)</p> <p>MARGINALIZE(Y)</p> <p>MARGINALIZE(Z)</p> <p>OBSERVE(z)</p>	<p>Named <code>delay_triplet</code> in supplementary material.</p> <p>No MARGINALIZE(X) is necessary: X, as a root node, is initialized in the marginalized state.</p>
<pre>print(x);</pre>	value(X)	<p>SAMPLE(Y)</p> <p>SAMPLE(X)</p>	<p>Samples $Y \sim p(dy z)$.</p> <p>Samples $X \sim p(dx y, z)$.</p>
<pre>print(y);</pre>			A value $Y = y$ is already known.
<pre>x ~ N(0,1); for (t in 1..T) { <u>y[t]</u> ~ N(x,1); }</pre>	<p>assume(X)</p> <p>observe(y_t)</p>	<p>INITIALIZE(X)</p> <p>INITIALIZE(y_t)</p> <p>MARGINALIZE(y_t)</p> <p>OBSERVE(y_t)</p>	<p>Named <code>delay_iid</code> in supplementary material. It encodes multiple i.i.d. observations with a conjugate prior distribution over their mean.</p>
<pre>print(x);</pre>	value(X)	SAMPLE(X)	Samples $X \sim p(dx y_1, \dots, y_T)$.
<pre>x ~ Bernoulli(p); if (x) { y ~ N(0,1); } else { y <- 0; }</pre>	<p>assume(X)</p> <p>value(X)</p> <p>assume(Y)</p>	<p>INITIALIZE(X)</p> <p>SAMPLE(X)</p> <p>INITIALIZE(Y)</p>	<p>Named <code>delay_spike_and_slab</code> in supplementary material. It encodes a spike-and-slab prior [15] often used in Bayesian linear regression.</p> <p>Used as a regular variable, no graph operations are triggered.</p> <p>Y is marginalized or realized as some $Y = y$ by the end, according to the stochastic branch.</p>
<pre>x[1] ~ N(0,1); <u>y[1]</u> ~ N(x[1],1);</pre>	<p>assume(X_1)</p> <p>observe(y_1)</p>	<p>INITIALIZE(X_1)</p> <p>INITIALIZE(y_1)</p> <p>MARGINALIZE(y_1)</p> <p>OBSERVE(y_1)</p>	<p>Named <code>delay_kalman</code> in supplementary material. It encodes a linear-Gaussian state-space model, for which delayed sampling yields a forward Kalman filter and backward simulation.</p>
<pre>for (t in 2..T) { x[t] ~ N(a*x[t-1],1); <u>y[t]</u> ~ N(x[t],1); }</pre>	<p>assume(X_t)</p> <p>observe(y_t)</p>	<p>INITIALIZE(X_t)</p> <p>INITIALIZE(y_t)</p> <p>MARGINALIZE(X_t)</p> <p>MARGINALIZE(y_t)</p> <p>OBSERVE(y_t)</p>	<p>After each tth iteration of this loop, the distribution $p(dx_t y_1, \dots, y_t)$ is obtained; the behavior corresponds to a Kalman filter.</p>
<pre>print(x[1]);</pre>	value(X_1)	<p>SAMPLE(X_T)</p> <p>...</p> <p>SAMPLE(X_1)</p>	<p>Samples $X_T \sim p(dx_T y_1, \dots, y_T)$.</p> <p>Recursively samples $X_t \sim p(dx_t x_{t+1}, y_1, \dots, y_t)$ and computes $p(dx_{t-1} x_t, y_1, \dots, y_{t-1})$.</p> <p>Samples $X_1 \sim p(dx_1 x_2, y_1)$.</p>

Table 1: Pedagogical examples of delayed sampling applied to four probabilistic programs, showing the programs themselves (first column), the checkpoints reached as they execute linearly from top to bottom (second column), the sequence of local operations that these trigger on the graph (third column), and commentary (fourth column). The programs use a Birch-like syntax. Random variables with given values (from earlier assignment) are annotated by underlining. The function `print` is assumed to accept real-valued arguments only, so may trigger a value checkpoint when used.

```

INITIALIZE( $v, p(\cdot)$ )
1  if  $p$  includes a parent node,  $u$ 
2      $I \leftarrow I \cup \{v\}$ 
3      $E \leftarrow E \cup \{(u, v)\}$ 
4      $q_v(dx_v) \leftarrow p(dx_v \mid x_u)$ 
5  else
6      $M \leftarrow M \cup \{v\}$ 
7      $q_v(dx_v) \leftarrow p(dx_v)$ 

MARGINALIZE( $v$ )
1  assert  $v \in I$  and  $v$  has a parent  $u \in T$ 
2   $q_v(dx_v) \leftarrow \int_{x_u} p(dx_v \mid x_u) q_u(dx_u)$ 
3   $I \leftarrow I \setminus \{v\}$ 
4   $M \leftarrow M \cup \{v\}$ 

SAMPLE( $v$ )
1  assert  $v \in T$ 
2  draw  $x_v \sim q_v(dx_v)$ 
3  REALIZE( $v$ )

OBSERVE( $v$ )
1  assert  $v \in T$ 
2   $w \leftarrow q_v(x_v)w$ 
3  REALIZE( $v$ )

REALIZE( $v$ )
1  assert  $v \in T$ 
2   $M \leftarrow M \setminus \{v\}$ 
3   $R \leftarrow R \cup \{v\}$ 
4  if  $v$  has a parent  $u$  // condition parent
5      $q_u(dx_u) \leftarrow \frac{p(x_v \mid x_u) q_u(dx_u)}{\int_{x_u} p(x_v \mid x_u') q_u(dx_u')}$ 
6      $E \leftarrow E \setminus \{(u, v)\}$ 
7  for  $u \in \text{Ch}(v)$  // new roots from children
8     MARGINALIZE( $u$ )
9      $E \leftarrow E \setminus \{(v, u)\}$ 

GRAFT( $v$ )
1  if  $v \in M$ 
2     if  $v$  has a child  $u \in M$ 
3         PRUNE( $u$ )
4  else
5     GRAFT( $u$ ) where  $u$  is the parent of  $v$ 
6     MARGINALIZE( $v$ )
7  assert  $v \in T$ 

PRUNE( $v$ )
1  assert  $v \in M$ 
2  if  $v$  has a child  $u \in M$ 
3     PRUNE( $u$ )
4  SAMPLE( $v$ )
    
```

Figure 1: Operations on the graph. The left arrow (\leftarrow) denotes assignment. Assigning to a distribution is interpreted as updating its hyperparameters.

given in Appendices C and D.

Table 1 provides pedagogical examples using a Birch-like syntax, showing the sequence of checkpoints and graph operations triggered as some simple programs execute. They show how delayed sampling behaves through programming structures such as conditionals and loops, including stochastic branches.

In addition, we provide two case studies where delayed sampling improves inference, firstly a linear-nonlinear state-space model with simulated data, secondly a vector-borne disease model with real data from an outbreak of dengue virus in Micronesia. We use a simple random-wegue or pseudo-marginal-type importance sampling algorithm for both of these examples:

1. Run SMC on the probabilistic program with delayed sampling enabled, producing N number of samples x^1, \dots, x^N with associated weights w^1, \dots, w^N and a marginal likelihood estimate \hat{Z} .
2. Draw $a \in \{1, \dots, N\}$ from the categorical distribution defined by $P(a) = w^a / \sum_{n=1}^N w^n$.
3. Output x^a with weight \hat{Z} .

This produces one sample with associated weight, but may be repeated as many times as necessary—in parallel, even—to produce an importance sample as large as desired. The success of the approach depends on the variance of \hat{Z} . This variance can be reduced by marginalizing out one or more variables (recall Section 2.1). This is what delayed sampling achieves, and so we compare the variance of \hat{Z} with delayed sampling enabled and disabled. When disabled, the SMC algorithm is simply a bootstrap particle filter. When enabled, it yields a Rao–Blackwellized particle filter. Where parameters are involved (as in the second case study), the diversity of parameter values depletes through the resampling step of SMC. This has motivated more sophisticated methods for parameter estimation such as particle Markov chain Monte Carlo methods [1], also applied to probabilistic programs [28]. Particle Gibbs is an obvious candidate here. We find, however, that the reduction in variance afforded by marginalizing out one or more variables with delayed sampling is sufficient to enable the above importance sampling algorithm for the two case studies here.

3.1 Linear-nonlinear state-space model

The first example is that of a mixed linear-nonlinear state-space model. For this model, delayed sampling yields a particle filter with locally-optimal proposal and Rao–Blackwellization.

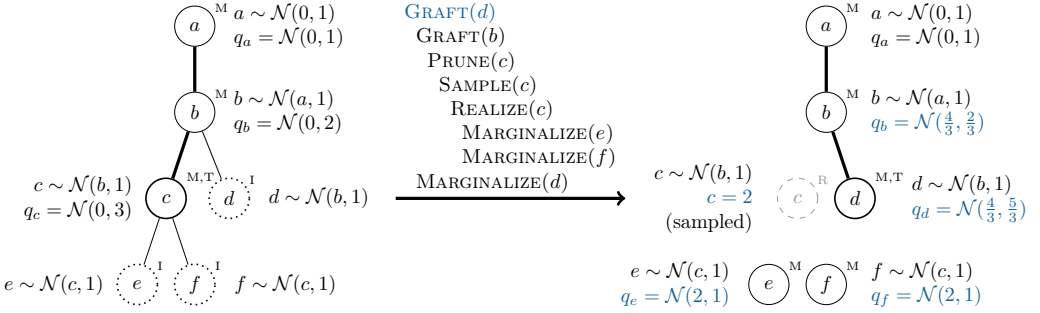


Figure 2: Demonstration of the M -path and operations on the graph. On the left, the M -path reaches from the root node, a , to the terminal node, c , marked in bold lines. The **GRAFT** operation is called for d . This requires a realignment of the M -path around b , pruning the previous M -path at c , then extending it through to d . The stack trace of operations is in the center, and the final state on the right. Descendants of c that were not on the M -path are now the roots of separate, disjoint trees.

The model is given by [12] and repeated in Appendix A. It consists of both nonlinear and linear-Gaussian state variables, as well as nonlinear and linear-Gaussian observations. Parameters are fixed. Ideally, the linear-Gaussian substructure is solved analytically (e.g. using a Kalman filter), leaving only the nonlinear substructure to sample (e.g. using a particle filter). The Rao-Blackwellized particle filter, also known as the marginalized particle filter, was designed to achieve precisely this [2, 22].

Delayed sampling automatically yields this method for this model, as long as analytical relationships between multivariate Gaussian distributions are encoded. In Birch these are implemented as supernodes: single nodes in the graph that contain multiple random variables. While the relationships between individual variables in a multivariate Gaussian have, in general, directed acyclic graph structure, their implementation as supernodes maintains the required tree structure.

The model is run for 100 time steps to simulate data. It is run again with SMC, conditioning on this data. For various numbers of particles, it is run 100 times to estimate \hat{Z} , with delayed sampling enabled and disabled. Figure 3 (left) plots the distribution of these estimates. Clearly, with delayed sampling enabled, fewer particles are needed to achieve comparable variance in the log-likelihood estimate.

3.2 Vector-borne disease model

The second example is an epidemiological case study of an outbreak of dengue virus: a mosquito-borne tropical disease with an estimated 50-100 million cases and

10000 deaths worldwide each year [24]. It is based on the study in [6], which jointly models two outbreaks of dengue virus and one of Zika virus in two separate locations (and populations) in Micronesia. Presented here is a simpler study limited to one of those outbreaks, specifically that of dengue on the Yap Main Islands in 2011. The data used consists of 172 observations of reported cases, on a daily basis during the main outbreak, and on a weekly basis before and after.

The model consists of two components, representing the human and mosquito populations, coupled via cross-infection. Each population is further divided into subpopulations of susceptible, exposed, infectious and recovered individuals. At each time step a binomial transfer occurs between subpopulations, parameterized with conjugate beta priors. Details are in Appendix B.

The task is both parameter and state estimation. For this model, delayed sampling produces a Rao-Blackwellized particle filter where parameters, rather than state variables, are marginalized out. While the state variables are sampled immediately, the parameters are maintained in a marginalized state, conditioned on the samples of these state variables. This is a consequence of conjugacy between the beta priors on parameters and the binomial likelihoods of the state variables (as pseudo-observations).

For various numbers of particles, SMC is run 100 times to estimate \hat{Z} , with delayed sampling enabled and disabled. Figure 3 (right) plots the distribution of these estimates. Clearly, with delayed sampling enabled, fewer particles are needed to achieve comparable variance in the log-likelihood estimate. Some posterior results are given in Appendix B.

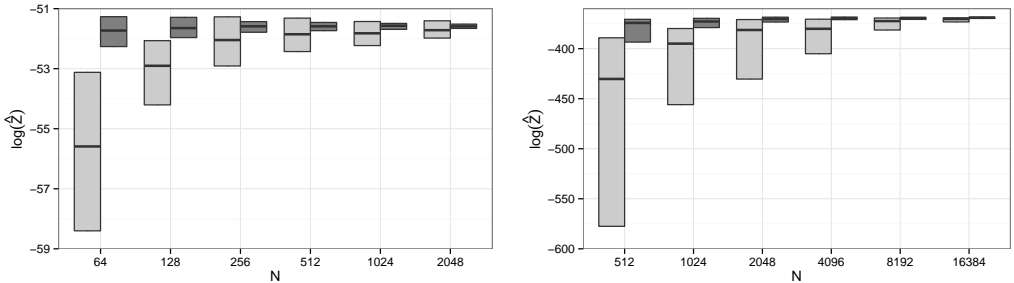


Figure 3: Distribution of the marginal log-likelihood estimate ($\log \hat{Z}$) for different numbers of particles (N) over 100 runs for (left) the linear-nonlinear state-space model, and (right) the vector-borne disease model, with (light gray) delayed sampling disabled, corresponding to a bootstrap particle filter, and (dark gray) delayed sampling enabled, corresponding to a Rao–Blackwellized particle filter. All runs use systematic resampling [10] when effective sample size falls below $0.7N$. Boxes indicate the interquartile range, midline the median. In both cases, significantly fewer particles are required to achieve comparable variance when delayed sampling is enabled.

4 DISCUSSION AND CONCLUSION

Table 1 demonstrates how delayed sampling operates through typical program structures such as conditionals and loops, including stochastic branches as encountered in probabilistic programs. Figure 3 demonstrates the potential gains. These are particularly encouraging given that the mechanism is mostly automatic.

Some limitations are worth noting. The graph of analytically-tractable relationships must be a forest of disjoint trees. It is unclear whether this is a significant limitation in practice, but support for more general structures may be desirable. It is worth emphasizing that this relates to the structure of analytically-tractable relationships and the ability of the mechanism to utilize them, not to the structure of the model as a whole. At present, for more general structures, some opportunities for variance reduction are missed. One remedy is to encode supernodes, as for the multivariate Gaussian distributions in Section 3.1.

Delayed sampling potentially reorders the sampling associated with assume checkpoints, and the interleaving of this amongst observe checkpoints, but does not reorder the execution of observe checkpoints. There is an opportunity cost to this. Consider the final example in Table 1: move the observations y_1, \dots, y_T into a second loop that traverses time backward from T to 1. Delayed sampling now draws each x_t from $p(dx_t | x_{t+1}, y_t)$, not $p(dx_t | x_{t+1}, y_1, \dots, y_t)$. This is suboptimal but not incorrect: whatever the distribution, importance weights correct for its discrepancy from the target. It is again unclear whether this is a significant limitation in practice; examples seem contrived and easily fixed by reordering code.

While delayed sampling may reduce the number of samples required for comparable variance, it does require additional computation per sample. For univariate relationships (e.g. beta-binomial, gamma-Poisson), this overhead is constant and—we conjecture—likely worthwhile for any fixed computational budget. For multivariate relationships the overhead is more complex and may not be worthwhile (e.g. multivariate Gaussian conjugacies require matrix inversions that are $\mathcal{O}(N^3)$ in the number of dimensions). A thorough empirical comparison is beyond the scope of this article.

Finally, while the focus of this work is SMC, delayed sampling may be useful in other contexts. With undirected graphical models, for example, delayed sampling may produce a collapsed Gibbs sampler. This is left to future work.

Acknowledgements

This research was financially supported by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE*. Jan Kudlicka was supported by the Swedish Research Council grant 2013-4853.

Supplementary material

Appendix A details the linear-nonlinear state-space model, and Appendix B the vector-borne disease model. Appendix C details the Anglican implementation, and Appendix D the Birch implementation. Code is included for the pedagogical examples in both Anglican and Birch, and for the empirical case studies, along with data sets, in Birch only.

References

- [1] C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society B*, 72:269–302, 2010. doi: 10.1111/j.1467-9868.2009.00736.x.
- [2] R. Chen and J. S. Liu. Mixture Kalman filters. *Journal of the Royal Statistical Society B*, 62:493–508, 2000.
- [3] P. Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer-Verlag, New York, 2004.
- [4] P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society B*, 68:441–436, 2006. doi: 10.1111/j.1467-9868.2006.00553.x.
- [5] A. Doucet and A. M. Johansen. *A tutorial on particle filtering and smoothing: fifteen years later*, chapter 24, pages 656–704. Oxford University Press, 2011.
- [6] S. Funk, A. J. Kucharski, A. Camacho, R. M. Eggo, L. Yakob, L. M. Murray, and W. J. Edmunds. Comparative analysis of dengue and Zika outbreaks reveals differences by setting and virus. *PLOS Neglected Tropical Diseases*, 10(12):1–16, 12 2016. doi: 10.1371/journal.pntd.0005173.
- [7] H. Ge, A. Ścibior, K. Xu, and Z. Ghahramani. Turing: A fast imperative probabilistic programming language. Technical report, June 2016.
- [8] N. D. Goodman and A. Stuhlmüller. The design and implementation of probabilistic programming languages. <http://dippl.org>, 2014.
- [9] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140:107–113, 1993. doi: 10.1049/ip-f-2.1993.0015.
- [10] G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25, 1996. doi: 10.2307/1390750.
- [11] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 1988.
- [12] F. Lindsten and T. B. Schön. Identification of mixed linear/nonlinear state-space models. In *49th IEEE Conference on Decision and Control (CDC)*, pages 6377–6382, 2010.
- [13] D. Lundén. Delayed sampling in the probabilistic programming language Anglican. Master’s thesis, KTH Royal Institute of Technology, School of Computer Science and Communication, 2017.
- [14] V. K. Mansinghka, D. Selsam, and Y. N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv abs/1404.0099*, 2014.
- [15] T. J. Mitchell and J. J. Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83:1023–1032, 1988. doi: 10.2307/2290129.
- [16] L. M. Murray. Bayesian state-space modelling on high-performance hardware using LibBi. *Journal of Statistical Software*, 67(10):1–36, 2015. doi: 10.18637/jss.v067.i10.
- [17] A. Nori, C.-K. Hur, S. Rajamani, and S. Samuel. R2: An efficient MCMC sampler for probabilistic programs. *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.
- [18] B. Paige and F. Wood. A compilation target for probabilistic programming languages. *31st International Conference on Machine Learning (ICML)*, 2014.
- [19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [20] A. Pfeffer. *Practical Probabilistic Programming*. Manning, 2016.
- [21] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, 2004. doi: 10.1007/978-1-4757-4145-2.
- [22] T. Schön, F. Gustafsson, and P. Nordlund. Marginalized particle filters for mixed linear/nonlinear state-space models. *IEEE Transactions on Signal Processing*, 53:2279–2289, 2005. doi: 10.1214/193940307000000518.
- [23] C. Shan and N. Ramsey. Exact Bayesian inference by symbolic disintegration. *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2017.
- [24] J. D. Stanaway, D. S. Shepard, E. A. Undurraga, Y. A. Halasa, L. E. Coffeng, O. J. Brady, S. I. Hay, N. Bedi, I. M. Bensenor, C. A. Castañeda Orjuela, T.-W. Chuang, K. B. Gibney, Z. A. Memish, A. Rafay, K. N. Ukwaja, N. Yonemoto, and C. J. L. Murray. The global burden of dengue: an analysis from the Global Burden of Disease Study 2013. *The Lancet Infectious Diseases*, 16(6):712–723, 2016. doi: 10.1016/s1473-3099(16)00026-8.

- [25] A. Todeschini, F. Caron, M. Fuentes, P. Legrand, and P. Del Moral. Biips: Software for Bayesian inference with interacting particle systems. *arXiv abs/1412.3779*, 2014.
- [26] D. Tolpin, J. van de Meent, H. Yang, and F. Wood. Design and implementation of probabilistic programming language Anglican. *arXiv abs/1608.05263*, 2016.
- [27] D. Wingate, A. Stuhmueller, and N. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. *14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 770–778, 2011.
- [28] F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

A Details of the linear-nonlinear state-space model

The full model is described in [12]. The state model contains both nonlinear (X_t^n) and linear-Gaussian (X_t^l) state variables, and is given by:

$$\begin{aligned} X_0^n &\sim \mathcal{N}(0, 1) \\ X_t^n &\sim \mathcal{N}(\arctan x_{t-1}^n + Bx_{t-1}^l, 0.01) \\ X_0^l &\sim \mathcal{N}(0, I_{3 \times 3}) \\ X_t^l &\sim \mathcal{N}(Ax_{t-1}^l, 0.01I_{3 \times 3}). \end{aligned}$$

The observation model contains both nonlinear (Y_t^n) and linear-Gaussian (Y_t^l) observations, and is given by:

$$\begin{aligned} Y_t^n &\sim \mathcal{N}(0.1(x_t^n)^2 \text{sgn}(x_t^n), 0.1) \\ Y_t^l &\sim \mathcal{N}(Cx_t^l, 0.1I_{3 \times 3}). \end{aligned}$$

Parameters are fixed as follows:

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0.3 & 0 \\ 0 & 0.92 & -0.3 \\ 0 & 0.3 & 0.92 \end{pmatrix} \\ B &= (1 \quad 0 \quad 0) \\ C &= (1 \quad -1 \quad 1). \end{aligned}$$

B Details of the vector-borne disease model

The process model is a discrete-time and discrete-state stochastic model based on the continuous-time and continuous-state deterministic mean-field approximation used in [6]. It consists of two SEIR (susceptible, exposed, infectious, recovered) compartmental models, one for the human population, the other for the mosquito population, coupled via cross-infection terms. Each component consists of state variables giving population counts in each of the four compartments: s (susceptible), e (exposed), i (infectious), and r (recovered), along with a total population n that maintains the identity $n = s + e + i + r$, and parameters ν (birth probability), μ (death probability), λ (transmission probability), δ (infectious probability), and γ (recovery probability). A susceptible human may become infected when bitten by an infectious mosquito, while a susceptible mosquito may become infected when biting an infectious human.

We use superscript h to denote state variables and parameters associated with the human component, and superscript m to denote those associated with the mosquito component. For state variables, subscripts index time in days.

B.1 Initial condition model

For the setting of Yap Main Islands in 2011, the following initial conditions are prescribed:

$$\begin{aligned} n_0^h &= 7370 & n_0^m &= 10^u n_0^h \\ s_0^h &= n_0^h - e_0^h - i_0^h - r_0^h & s_0^m &= n_0^m \\ e_0^h &\sim \text{Poisson}(10) & e_0^m &= 0 \\ i_0^h - 1 &\sim \text{Poisson}(10) & i_0^m &= 0 \\ r_0^h &\sim \text{Binomial}(n_1^h, 6/100) & r_0^m &= 0, \end{aligned}$$

with $u \sim \mathcal{U}(-1, 2)$.

B.2 Transition model

The model transitions in two steps. The first step is an exchange between compartments that preserves total population. Denoting with primes the intermediate state after this first step, we have:

$$\begin{aligned} s_t^{h'} &= s_{t-1}^h - \oplus e_t^h & s_t^{m'} &= s_{t-1}^m - \oplus e_t^m \\ e_t^{h'} &= e_{t-1}^h + \oplus e_t^h - \oplus i_t^h & e_t^{m'} &= e_{t-1}^m + \oplus e_t^m - \oplus i_t^m \\ i_t^{h'} &= i_{t-1}^h + \oplus i_t^h - \oplus r_t^h & i_t^{m'} &= i_{t-1}^m + \oplus i_t^m - \oplus r_t^m \\ r_t^{h'} &= r_{t-1}^h + \oplus r_t^h & r_t^{m'} &= r_{t-1}^m + \oplus r_t^m, \end{aligned}$$

with the newly exposed, infectious, and recovered populations distributed as:

$$\begin{aligned} \oplus e_t^h &\sim \text{Binomial}(\tau_t^h, \lambda^h) & \oplus e_t^m &\sim \text{Binomial}(\tau_t^m, \lambda^m) \\ \oplus i_t^h &\sim \text{Binomial}(e_{t-1}^h, \delta^h) & \oplus i_t^m &\sim \text{Binomial}(e_{t-1}^m, \delta^m) \\ \oplus r_t^h &\sim \text{Binomial}(i_{t-1}^h, \gamma^h) & \oplus r_t^m &\sim \text{Binomial}(i_{t-1}^m, \gamma^m), \end{aligned}$$

for parameters λ^h , δ^h , γ^h , λ^m , δ^m , γ^m . The τ_t^h gives the number of susceptible humans bitten by at least one infectious mosquito, and τ_t^m the number of susceptible mosquitos that bite at least one infectious human:

$$\tau_t^h \sim \text{Binomial}(s_{t-1}^h, 1 - \exp(-i_{t-1}^m/n_{t-1}^h)) \quad (4)$$

$$\tau_t^m \sim \text{Binomial}(s_{t-1}^m, 1 - \exp(-i_{t-1}^h/n_{t-1}^m)). \quad (5)$$

These latter quantities are derived by assuming (a) a Poisson(n_t^m) number of mosquito blood meals per day with these interactions uniformly distribution across both humans and mosquitos, (b) that a human is infected with probability λ^h if interacting one or more times with an infectious mosquito, and (c) that a mosquito is infected with probability λ^m if interacting one or more times with an infectious human. Note that the n_{t-1}^h appearing in (4) is correct, although one may expect to see n_{t-1}^m given the otherwise-symmetry of the equations of this model. In the derivation, n_{t-1}^m also appears in the denominator of both (4) and (5), but cancels with the Poisson rate parameter for the number of blood meals, also given by n_t^m as above.

The second step accounts for births and deaths:

$$\begin{aligned} s_t^h &= s_t^{h'} - \ominus s_t^h + \oplus n_t^h & s_t^m &= s_t^{m'} - \ominus s_t^m + \oplus n_t^m \\ e_t^h &= e_t^{h'} - \ominus e_t^h & e_t^m &= e_t^{m'} - \ominus e_t^m \\ i_t^h &= i_t^{h'} - \ominus i_t^h & i_t^m &= i_t^{m'} - \ominus i_t^m \\ r_t^h &= r_t^{h'} - \ominus r_t^h & r_t^m &= r_t^{m'} - \ominus r_t^m, \end{aligned}$$

with births distributed as

$$\oplus n_t^h \sim \text{Binomial}(n_t^{h'}, \nu^h) \quad \oplus n_t^m \sim \text{Binomial}(n_t^{m'}, \nu^m),$$

with parameters ν^h and ν^m , and deaths as

$$\begin{aligned} \ominus s_t^h &\sim \text{Binomial}(s_t^{h'}, \mu^h) & \ominus s_t^m &\sim \text{Binomial}(s_t^{m'}, \mu^m) \\ \ominus e_t^h &\sim \text{Binomial}(e_t^{h'}, \mu^h) & \ominus e_t^m &\sim \text{Binomial}(e_t^{m'}, \mu^m) \\ \ominus i_t^h &\sim \text{Binomial}(i_t^{h'}, \mu^h) & \ominus i_t^m &\sim \text{Binomial}(i_t^{m'}, \mu^m) \\ \ominus r_t^h &\sim \text{Binomial}(r_t^{h'}, \mu^h) & \ominus r_t^m &\sim \text{Binomial}(r_t^{m'}, \mu^m), \end{aligned}$$

with parameters μ^h and μ^m .

B.3 Observation model

Observations are of the number of new infectious cases reported at health centers, aggregated over the time since the last such observation (this is daily during the peak time of the outbreak and weekly either side). For times $t \in \{1, \dots, T\}$ where observations are available, the observation model is given by

$$y_t \sim \text{Binomial}\left(\sum_{s=t-l_t+1}^t \oplus i_s^h, \rho\right),$$

where l_t (lag) indicates the number of days since the last observation. Significant under-reporting of cases is expected, reflected in the parameter ρ .

B.4 Parameter model

The following fixed values and priors are assigned to parameters, translating prior knowledge on rates in [6] to prior knowledge on probabilities here:

$$\begin{aligned} \nu_h &= 0 & \nu_m &= 1/7 \\ \mu_h &= 0 & \mu_m &= 1/7 \\ \lambda_h &\sim \text{Beta}(1, 1) & \lambda_m &\sim \text{Beta}(1, 1) \\ \delta_h &\sim \text{Beta}\left(\frac{16}{11}, \frac{28}{11}\right) & \delta_m &\sim \text{Beta}\left(\frac{17}{13}, \frac{35}{13}\right) \\ \gamma_h &\sim \text{Beta}\left(\frac{13}{9}, \frac{23}{9}\right) & \gamma_m &= 0. \end{aligned}$$

Birth and death in the human population are assumed to be of minimal impact over the course of the outbreak, and so their rates are fixed to zero. The expected lifespan of a mosquito is one week, with birth and

death rates fixed accordingly. Mosquitos do not recover before death.

Finally, the prior over the reporting probability is

$$\rho \sim \text{Beta}(1, 1).$$

B.5 Inference results

Inference is performed by drawing 10000 weighted samples, each time running SMC with 8192 particles. The effective sample size of these 10000 weighted samples is computed to be 2260. Some results are shown in Figure 4.

C Anglican implementation

Anglican is a functional probabilistic programming language integrated with Clojure. Clojure, in turn, is a Lisp dialect which compiles to Java virtual machine bytecode, enabling reuse of the Java infrastructure. The Anglican compiler is built with Clojure macros, and compiles Anglican programs into continuation-passing-style Clojure code. This transformation enables inference algorithms to affect the control flow and record information at checkpoints. Manipulations are performed both on the continuations themselves and on the state, which is passed along as an argument in each continuation call.

For simplicity, delayed sampling is implemented entirely on top of the existing Anglican language, leaving the original language constructs and functionality untouched. A set of new keywords and functions are added for usage of delayed sampling: `ds-<name>`, `ds-value`, and `ds-observe`. The `ds-value` and `ds-observe` functions loosely correspond to the `SAMPLE` and `OBSERVE` operations in Section 2.2, but `ds-value` also includes functionality for retrieving values for already-sampled nodes. The set of `ds-<name>` functions correspond to the `INITIALIZE` operations in Section 2.2, for various probability distributions, e.g. `ds-normal`. The delayed sampling graph is conveniently encoded in the already existing Anglican state.

As an example, consider the following line of code:

```
let [x (ds-normal mean sd)]
```

This binds `x` to a graph node which is normally distributed with mean `mean` and standard deviation `sd`. To subsequently introduce another normally distributed graph node with the node `x` as mean, one can write

```
let [y (ds-normal x sd')]
```

passing the previous graph node `x` as a parameter. This will initialize a conjugate prior relationship between

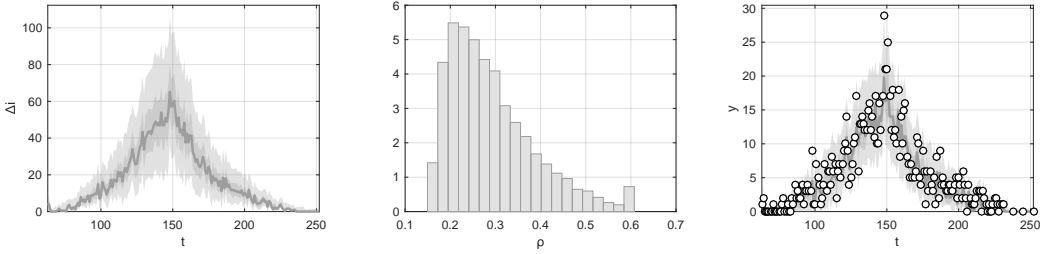


Figure 4: Posterior results for the vector-borne disease model example, (left) posterior distribution of newly infectious cases in humans over time, $\oplus i_t^h$, (middle) posterior distribution of the reporting probability parameter, ρ , and (right) posterior predictive distribution of the number of reported cases, y , overlaid with actual observations. In the left and right plots, the bold line gives the median, darker shaded region the 50% credibility interval, and lighter shaded region the 95% credibility interval.

them. If y is then observed, x will be conditioned on the observed value of y .

D Birch implementation

Birch is a compiled, imperative, object-oriented, generic, and probabilistic programming language. The latter is its primary research concern. The Birch compiler uses C++ as a target language.

Delayed sampling has been implemented using the Birch type system. Special types are used when declaring variables to make them eligible for delayed sampling. For example, a variable that might ordinarily be declared to be of type `Real` may be declared to be of type `Random<Real>` to make it eligible for delayed sampling. The generic class `Random` implements the behavior required for delayed sampling, and is specialized into classes that encode distributions (e.g. `Gaussian`), then further into classes that encode distributions with analytical relationships to others (e.g. `GaussianWithGaussianMean`). The graph required for delayed sampling is formed implicitly through objects of these classes and their member attributes.

Birch supports implicit type conversion, compiling directly to the same feature in C++. These implicit conversions are used to automatically trigger the value checkpoint, and are resolved at compile time. For example, a `Random<Real>` object may be passed to a function that requires a `Real` argument. An implicit conversion is used to trigger a value checkpoint, realizing a value of type `Real` from the object of type `Random<Real>`. In this way, the programmer need not explicitly indicate value checkpoints.

Paper IV



IEEE Copyright Notice

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Accepted to be published in

Proceedings of ICASSP 2020, IEEE International Conference on Acoustics, Speech and Signal Processing, Barcelona, Spain, May 4–8, 2020.

Please cite this version:

```
@inproceedings{,
  title={Particle filter with rejection control and unbiased estimator of
        the marginal likelihood},
  author={Kudlicka, Jan and Murray, Lawrence M. and Sch\"on, Thomas B. and
        Lindsten, Fredrik},
  booktitle={{IEEE} International Conference on Acoustics, Speech and Signal Processing,
            {ICASSP} 2020, Barcelona, Spain, May 4-8, 2020},
  publisher={{IEEE}},
  year={2020}
}
```

Note that the accepted version does not include the appendices that are included in the extended version (this document).

PARTICLE FILTER WITH REJECTION CONTROL AND UNBIASED ESTIMATOR OF THE MARGINAL LIKELIHOOD

Jan Kudlicka* Lawrence M. Murray† Thomas B. Schön* Fredrik Lindsten‡

* Department of Information Technology, Uppsala University, Uppsala, Sweden

† Uber AI, San Francisco, CA, USA

‡ Division of Statistics and Machine Learning, Linköping University, Linköping, Sweden

ABSTRACT

We consider the combined use of resampling and partial rejection control in sequential Monte Carlo methods, also known as particle filters. While the variance reducing properties of rejection control are known, there has not been (to the best of our knowledge) any work on unbiased estimation of the marginal likelihood (also known as the model evidence or the normalizing constant) in this type of particle filter. Being able to estimate the marginal likelihood without bias is highly relevant for model comparison, computation of interpretable and reliable confidence intervals, and in *exact approximation* methods, such as particle Markov chain Monte Carlo. In the paper we present a particle filter with rejection control that enables unbiased estimation of the marginal likelihood.

Index Terms— Particle filters, sequential Monte Carlo (SMC), partial rejection control, unbiased estimate of the marginal likelihood

1. INTRODUCTION

Rejuvenation of particles in methods based on sequential importance sampling is a crucial step to avoid the weight degeneracy problem. Sequential Monte Carlo (SMC) methods typically use *resampling*, but there are alternative methods available. *Rejection control*, proposed by Liu et al. [1] solves the degeneracy problem by checking the weights of particles (or *streams* in their terminology) at given checkpoints, and comparing them to given thresholds. Particles with weights below a certain checkpoint threshold are probabilistically discarded and replaced by new particles that are restarted from the beginning. Discarding particles that have passed through all previous checkpoints is quite disadvantageous. Liu proposed a modified version of the algorithm in [2], called *partial rejection control*, where a set of particles is propagated in parallel between the checkpoints, and each rejected particle gets replaced by a sample drawn from the particle set at the

previous checkpoint (quite similar to resampling), and propagated forward, rather than restarting from the beginning.

Peters et al. [3] combined partial rejection control and resampling: the resampling, propagation and weighting steps are the same as in standard SMC methods, but an additional step is performed after the weighting step. Here, the weight of each particle is compared to a threshold and if it falls below this threshold, the particle is probabilistically rejected, and the resampling, propagation and resampling steps are repeated. This procedure is repeated until the particle gets accepted. Peters et al. also adapted the algorithm to be used in an approximate Bayesian computation (ABC) setting.

We consider models with likelihoods and our contribution is a non-trivial modification of the particle filter with rejection control, allowing us to define an unbiased estimator of the marginal likelihood. This modification is very simple to implement: it requires an additional particle and counting the number of propagations. The unbiasedness of the marginal likelihood estimator opens for using particle filters with rejection control in exact approximate methods such as particle marginal Metropolis-Hastings (PMMH, [4]), model comparison, and computation of interpretable and reliable confidence intervals.

2. BACKGROUND

2.1. State space model

State space models are frequently used to model dynamical systems where the state evolution exhibits the Markov property (i.e., the state at time t only depends on the state at time $t-1$ but not on the state at any earlier time). Further, the state is not observed directly, but rather via measurements depending (stochastically) only on the state at the same time.

Let x_t denote the state at time t and y_t the corresponding measurement. The state space model can be represented using probability distributions:

$$x_0 \sim \mu_0(\cdot), \quad x_t \sim f_t(\cdot|x_{t-1}), \quad y_t \sim g_t(\cdot|x_t).$$

The inference goal is to estimate posterior distributions of (a subset of) the state variables given a set of measurements,

This research was supported by the Swedish Foundation for Strategic Research via the project ASSEMBLE (contract number: RIT15-0012) and by the Swedish Research Council grants 2013-4853 and 2017-03807.

Algorithm 1 Bootstrap particle filter (BPF)

```

 $\widehat{Z}_{\text{BPF}} \leftarrow 1$ 
for  $n = 1$  to  $N$  do                                 $\triangleright$  Initialize
   $x_0^{(n)} \sim \mu_0, w_0^{(n)} \leftarrow 1$ 
end for
for  $t = 1$  to  $T$  do
  for  $n = 1$  to  $N$  do
     $a^{(n)} \sim \mathcal{C}(\{w_{t-1}^{(m)}\}_{m=1}^N)$            $\triangleright$  Resample
     $x_t^{(n)} \sim f_t(\cdot | x_{t-1}^{(a^{(n)})})$          $\triangleright$  Propagate
     $w_t^{(n)} \leftarrow g_t(y_t | x_t^{(n)})$          $\triangleright$  Weight
  end for
   $\widehat{Z}_{\text{BPF}} \leftarrow \widehat{Z}_{\text{BPF}} \sum_{n=1}^N w_t^{(n)} / N$ 
end for

```

and to estimate the expected value of test functions with respect to these distributions. We are usually interested in the joint filtering distribution $p(x_{0:t}|y_{1:t})$ and the filtering distribution $p(x_t|y_{1:t})$, where $x_{0:t}$ denotes the sequence of all states until time t , i.e. x_0, x_1, \dots, x_t , and similarly for $y_{1:t}$.

2.2. Particle filters

Particle filters are sampling-based methods that construct sets \mathcal{S}_t of N weighted samples (particles) to estimate the filtering distribution $p(x_t|y_{1:t})$ for each time t . The baseline *bootstrap particle filter* creates the initial set \mathcal{S}_0 by drawing N samples from μ_0 and setting their (unnormalized) weights to 1, i.e.,

$$\mathcal{S}_0 = \left\{ (x_0^{(n)}, w_0^{(n)}) \mid x_0^{(n)} \sim \mu_0, w_0^{(n)} = 1 \right\}_{n=1}^N.$$

The set \mathcal{S}_t at time t is constructed from the previous set of particles \mathcal{S}_{t-1} by repeatedly (N times) choosing a particle from \mathcal{S}_{t-1} with probabilities proportional to their weights (this step is called *resampling*), drawing a new sample x from $f_t(\cdot | x_{t-1}^{(a)})$ where a is the index of the chosen particle (*propagation*), and setting its weight to $g_t(y_t | x)$ (*weighting*):

$$\mathcal{S}_t = \left\{ (x_t^{(n)}, w_t^{(n)}) \mid \begin{aligned} a_n &\sim \mathcal{C}(\{w_{t-1}^{(m)}\}_{m=1}^N) \\ x_t^{(n)} &\sim f_t(\cdot | x_{t-1}^{(a_n)}), \\ w_t^{(n)} &= g_t(y_t | x_t^{(n)}) \end{aligned} \right\}_{n=1}^N.$$

Here, \mathcal{C} denotes the categorical distribution with the unnormalized event probabilities specified as the parameter. The crucial element of a particle filter is the resampling step that avoids the weight degeneracy problem of sequential importance sampling. The pseudocode for the bootstrap particle filter is listed as Algorithm 1.

The unbiased estimator \widehat{Z}_{BPF} of the marginal likelihood $p(y_{1:T})$ (unbiased in the sense that $\mathbb{E}[\widehat{Z}_{\text{BPF}}] = p(y_{1:T})$) is

given by [5]:

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

Particle filters are a family of different variants of this algorithm. These variants use different proposal distributions to choose the initial set of samples, to propagate or to resample particles and use appropriate changes in the calculation of the importance weights w.r.t. the filtering distributions. In order to reduce the variance of estimators, some methods do not resample at every time step, but rather only when a summary statistic of weights crosses a given threshold, e.g., when the effective sample size (ESS) falls below νN , where $\nu \in [0, 1]$ is a tuning parameter.

In probabilistic programming, the state space model can be used to model program execution and particle filters are used as one of the general probabilistic programming inference methods. Examples of probabilistic programming languages that use particle filters and SMC for inference include Anglican [6], Biips [7], Birch [8], Figaro [9], LibBi [10], Venture [11], WebPPL [12] and Turing [13].

3. PARTICLE FILTER WITH REJECTION CONTROL

In certain models, such as models with jump processes or rare-event processes, or when the measurements contain outliers, the weights of many particles after propagation and weighting might be rather low or even zero. This decreases the ESS and thus increases the variance of the estimators of interest.

Below we present the *particle filter with rejection control* (PF-RC) that ensures that the weights of all particles in the particle set \mathcal{S}_t are greater than or equal to a chosen threshold, denoted by $c_t > 0$. The process of drawing new particles in PF-RC is almost identical to the process for the bootstrap particle filter described in the previous section, with one additional step that we will refer to as the *acceptance* step, described in the next paragraph.

Let $(x_t^{r(n)}, w_t^{r(n)})$ denote the particle after the resampling, propagation and weighting steps. The particle is accepted (and added to \mathcal{S}_t) with probability $\min(1, w_t^{r(n)}/c_t)$. If accepted, the particle weight is lifted to $\max(w_t^{r(n)}, c_t)$. If the particle is rejected, the resampling, propagation, weighting and acceptance steps are repeated until acceptance. The following table summarizes the acceptance step:

Condition	Acc. prob.	If accepted	If rejected
$w_t^{r(n)} \geq c_t$	1	$w_t^{(n)} \leftarrow w_t^{r(n)}$ $x_t^{(n)} \leftarrow x_t^{r(n)}$	—
$w_t^{r(n)} < c_t$	$w_t^{r(n)}/c_t$	$w_t^{(n)} \leftarrow c_t$ $x_t^{(n)} \leftarrow x_t^{r(n)}$	Sample new $x_t^{r(n)}$

Algorithm 2 Particle filter with rejection control (PF-RC)

```

 $\widehat{Z} \leftarrow 1$ 
for  $n = 1$  to  $N$  do                                ▷ Initialize
   $x_0^{(n)} \sim \mu_0, w_0^{(n)} \leftarrow 1$ 
end for
for  $t = 1$  to  $T$  do
   $P_t \leftarrow 0$ 
  for  $n = 1$  to  $N$  do
    repeat
       $a_t^{(n)} \sim \mathcal{C}(\{w_{t-1}^{(m)}\}_{m=1}^N)$                 ▷ Resample
       $x_t^{(n)} \sim f_t(\cdot | x_{t-1}^{(a_t^{(n)})})$                 ▷ Propagate
       $w_t^{(n)} \leftarrow g_t(y_t | x_t^{(n)})$                 ▷ Weight
       $P_t \leftarrow P_t + 1$ 
       $\alpha \sim \text{Bernoulli}(\min(1, w_t^{(n)} / c_t))$ 
    until  $\alpha$                                            ▷ Accept / reject
     $w_t^{(n)} \leftarrow \max(w_t^{(n)}, c_t)$                  ▷ Update the weight
  end for
  repeat                                               ▷ Additional particle
     $a' \sim \mathcal{C}(\{w_{t-1}^{(m)}\}_{m=1}^N)$ 
     $x' \sim f_t(\cdot | x_{t-1}^{(a')})$ 
     $w' \leftarrow g_t(y_t | x')$ 
     $P_t \leftarrow P_t + 1$ 
     $\alpha \sim \text{Bernoulli}(\min(1, w' / c_t))$ 
  until  $\alpha$ 
   $\widehat{Z} \leftarrow \widehat{Z} (\sum_{n=1}^N w_t^{(n)}) / (P_t - 1)$ 
end for

```

An important difference compared to the bootstrap particle filter is that we also use the same procedure (i.e., the resampling, propagation and acceptance steps) to sample one additional particle. This particle is *not* added to the particle set and its weight is not used either, but the number of propagations until its acceptance is relevant to the estimation of the marginal likelihood $p(y_{1:T})$.

Let P_t denote the total number of propagation steps performed in order to construct S_t as well as the additional particle. By the total number we mean that the propagation steps for both rejected and accepted particles are counted. The estimate \widehat{Z} of the marginal likelihood $p(y_{1:T})$ is given by

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

Theorem. *The marginal likelihood estimator \widehat{Z} is unbiased in sense that $\mathbb{E}[\widehat{Z}] = p(y_{1:T})$.*

Proof. See Appendix A. \square

The pseudocode for the PF-RC is listed as Algorithm 2. The gray color marks the part that is the same as in the bootstrap particle filter.

The question remains of how to choose the thresholds $\{c_t\}$. One option is to use some prior knowledge (that can

Table 1. Comparison of the filters for the model with outliers. See also the description in the text.

	c	N	ρ	ESS	ESS / ρ	var $\log \widehat{Z}$	ρ var $\log \widehat{Z}$
BPF	1024	1.00	101.6	101.6	2.18	2.18	
PF-RC	10^{-14}		1.06	180.1	169.8	1.13	1.20
	10^{-13}		1.08	285.8	264.0	1.08	1.17
	10^{-12}		1.12	386.1	346.1	1.02	1.14
	10^{-11}	1024	1.17	460.2	394.8	0.90	1.05
	10^{-10}		1.25	471.0	377.9	0.87	1.08
	10^{-9}		1.38	491.9	356.3	0.76	1.04
BPF	1200	1.17	185.6	158.3	1.91	2.24	

be obtained by pilot runs of a particle filter) and choose fixed thresholds. Liu et al. [1] mention several options for determining the thresholds dynamically after propagating and weighting all particles for the first time at each time step, using a certain quantile of these weights or a weighted average of the minimum, average and maximum weight, i.e., $c_t = p_1 \min w'_t + p_2 \bar{w}'_t + p_3 \max w'_t$, where all $p_i > 0$ and $p_1 + p_2 + p_3 = 1$. In general, setting the thresholds dynamically in each run breaks the unbiasedness of the marginal likelihood estimator (as demonstrated by an example in Appendix B).

Note that the alive particle filter [14] can be obtained as a limiting case of the particle filter with rejection control when all $c_t \rightarrow 0$. Instead of $\alpha \sim \text{Bernoulli}(\min(1, w/c_t))$ we need to use $\alpha \leftarrow \text{true}$ if $w > 0$ and false otherwise, but the rest of the algorithm remains the same.

4. EXPERIMENTS

4.1. Linear Gaussian state space model with outliers

The particle filter with rejection control may be useful in situations where measurements include outliers. To demonstrate this we considered the following linear Gaussian state space model:

$$x_0 \sim \mathcal{N}(0, 0.25), \quad x_t \sim \mathcal{N}(0.8x_{t-1}, 0.25),$$

$$y_t \sim \mathcal{N}(x_t, 0.1).$$

We simulated a set of measurements with outliers by replacing the measurement equation with $y_t \sim 0.9\mathcal{N}(x_t, 0.1) + 0.1\mathcal{N}(0, 1)$, and used these measurements in a set of experiment with both the BPF and a set of PF-RC with the thresholds at all checkpoints equal to a given value, i.e. $c_t = c$, where $c \in \{10^{-14}, 10^{-13}, \dots, 10^{-8}\}$. We ran each filter $M = 1000$ times using $N = 1024$ particles, collected a set of the estimates $\{\widehat{Z}_m\}_{m=1}^M$ of the marginal likelihood, and calculated several summary statistics, presented in Table 1. Here, the effective sample size ESS means

Table 2. Comparison of the filters for the object tracking problem.

Percentile	N	ρ	ESS	ESS / ρ	var $\log \hat{Z}$	ρ var $\log \hat{Z}$	
BPF	4096	1.00	8.5	8.5	482.00	482.00	
PF-RC	4096	50	2.68	4.3	1.6	94.75	253.68
		60	2.13	6.8	3.2	44.58	94.83
		70	1.99	8.7	4.4	35.22	70.18
		80	2.35	10.1	4.3	32.30	75.95
		90	3.39	13.7	4.0	14.63	49.54
		95	4.78	32.0	6.7	3.95	18.87
99	7.97	44.6	5.6	1.08	8.65		
BPF	32650	7.97	10.7	1.3	14.39	114.67	

$(\sum_{m=1}^M \hat{Z}_m)^2 / \sum_{m=1}^M \hat{Z}_m^2$, and ρ denotes the average number of propagations relatively to the number of propagation in the bootstrap particle filter with the same number of particles.

The filter that maximized ESS/ ρ used 1.17 times more propagations than the baseline BPF, so we also repeated the experiment using a BPF with 1200 particles to match the number of propagations; the results are shown in the last row.

4.2. Object tracking

We implemented the particle filter with rejection control as an inference method in the probabilistic programming language Birch [8]. We used the multiple object tracking model from [8] but restricted it to two objects that both appear at the initial time within the target area for computational purposes.

We used the program to simulate the tracks and measurements for 50 time steps. We ran experiments using 50-th, 60-th, 70-th, 80-th, 90-th, 95-th and 99-th percentiles as the thresholds, but in order to keep the marginal likelihood estimates unbiased, we first ran the program once for each of the percentiles (using 32768 particles) and saved the threshold values. We then ran the program $M = 100$ times for each of the percentiles, using the saved values as fixed thresholds and with only $N = 4096$ particles. In case where the threshold was 0, we fell back to accepting all particles. We compared the marginal likelihood estimates with the estimates obtained by running the program with the BPF inference. The results are summarized in Table 2 and Fig. 1.

We also repeated the experiments using a bootstrap particle filter with 32650 particles (to match the number of propagations in the filter with the 99-th percentile), the results are presented in the last row of the table.

5. DISCUSSION AND CONCLUSION

In this paper we presented a particle filter with rejection control (PF-RC) that enables unbiased estimation of the marginal likelihood. We briefly mentioned several situations where the

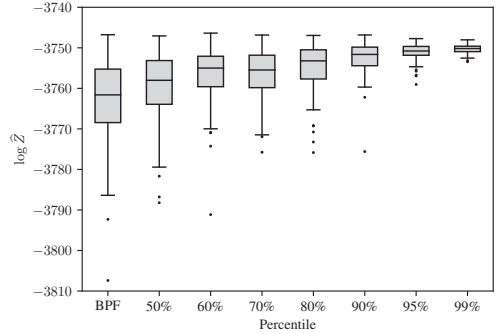


Fig. 1. Box plot of $\log \hat{Z}$ for the object tracking problem.

unbiasedness is important. We also showed a couple of examples that demonstrated the potential of the method. As we saw, the PF-RC outperformed (in terms of ESS and var $\log \hat{Z}$) the bootstrap particle filter (BPF) even when the latter used more particles and matched the total number of propagations. This is due to the fact that the number of propagations varies between the time steps in PF-RC, and more propagations are used “where it is needed”, while BPF uses the same number of propagations at each time step. The PF-RC also needs to use less memory compared to the BPF with the matched number of propagations, which might be an important advantage in problems requiring many particles. On the other hand, it might not always be clear how to determine the thresholds. In our future work we wish to look into this question, especially in the context of using PF-RC in exact approximate methods such as particle marginal Metropolis-Hastings method.

6. REFERENCES

- [1] Jun S Liu, Rong Chen, and Wing Hung Wong, “Rejection control and sequential importance sampling,” *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1022–1031, 1998.
- [2] Jun S Liu, *Monte Carlo strategies in scientific computing*, Springer Science & Business Media, 2008.
- [3] Gareth W Peters, Yanan Fan, and Scott A Sisson, “On sequential Monte Carlo, partial rejection control and approximate Bayesian computation,” *Statistics and Computing*, vol. 22, no. 6, pp. 1209–1222, 2012.
- [4] C. Andrieu, A. Doucet, and R. Holenstein, “Particle Markov chain Monte Carlo methods,” *Journal of the Royal Statistical Society: Series B*, vol. 72, no. 3, pp. 269–342, 2010.

- [5] Pierre Del Moral, *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*, Probability and Its Applications. Springer New York, 2004.
- [6] David Tolpin, Jan Willem van de Meent, Hongseok Yang, and Frank Wood, “Design and implementation of probabilistic programming language Anglican,” *arXiv preprint arXiv:1608.05263*, 2016.
- [7] Adrien Todeschini, François Caron, Marc Fuentes, Pier-rick Legrand, and Pierre Del Moral, “Biips: Software for Bayesian inference with interacting particle systems,” *arXiv preprint arXiv:1412.3779*, 2014.
- [8] Lawrence M Murray and Thomas B Schön, “Automated learning with a probabilistic programming language: Birch,” *Annual Reviews in Control*, vol. 46, pp. 29–43, 2018.
- [9] Avi Pfeffer, *Practical probabilistic programming*, Manning Publications, 2016.
- [10] Lawrence M Murray, “Bayesian state-space modelling on high-performance hardware using LibBi,” *Journal of Statistical Software*, vol. 67, no. 10, pp. 1–36, 2015.
- [11] Vikash Mansinghka, Daniel Selsam, and Yura Perov, “Venture: a higher-order probabilistic programming platform with programmable inference,” *arXiv preprint arXiv:1404.0099*, 2014.
- [12] Noah D Goodman and Andreas Stuhlmüller, “The design and implementation of probabilistic programming languages,” <http://dippl.org>, 2014, Accessed: 2019-10-21.
- [13] Hong Ge, Kai Xu, and Zoubin Ghahramani, “Tur-ing: a language for flexible probabilistic inference,” in *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, 2018, pp. 1682–1690.
- [14] Jan Kudlicka, Lawrence M Murray, Fredrik Ronquist, and Thomas B Schön, “Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

A. PROOF OF UNBIASEDNESS OF THE MARGINAL LIKELIHOOD ESTIMATOR

The proof follows the proof of unbiasedness of the marginal likelihood estimator for the alive particle filter given in [14].

Lemma 1.

$$\mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1} \middle| \mathcal{S}_{t-1} \right] = \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p \left(y_t \middle| x_{t-1}^{(n)} \right).$$

Proof. For brevity we omit conditioning on \mathcal{S}_{t-1} in the notation. A candidate sample x' is constructed by drawing a sample from \mathcal{S}_{t-1} with the probabilities proportional to the weights $\{w_{t-1}^{(n)}\}$ and propagating it forward to time t , i.e.

$$x' \sim \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t \left(x' \middle| x_{t-1}^{(n)} \right).$$

The candidate sample x' is accepted with probability $\min(1, g_t(y_t|x')/c_t)$. If the sample is rejected, a new candidate sample is drawn from the above-mentioned distribution. The acceptance probability p_{A_t} is given by

$$p_{A_t} = \int \min \left(1, \frac{g_t(y_t|x')}{c_t} \right) \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t \left(x' \middle| x_{t-1}^{(n)} \right) dx'.$$

Accepted samples are distributed according to the following distribution:

$$x_t \sim \frac{1}{p_{A_t}} \min \left(1, \frac{g_t(y_t|x_t)}{c_t} \right) \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t \left(x_t \middle| x_{t-1}^{(n)} \right)$$

and the expected value of the weight $w_t = \max(g_t(y_t|x_t), c_t)$ of an accepted sample is given by

$$\mathbb{E}[w_t] = \int \max(g_t(y_t|x_t), c_t) \frac{1}{p_{A_t}} \min \left(1, \frac{g_t(y_t|x_t)}{c_t} \right) \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t \left(x_t \middle| x_{t-1}^{(n)} \right) dx_t.$$

Note that $\max(g_t(y_t|x_t), c_t) \times \min(1, g_t(y_t|x_t)/c_t) = g_t(y_t|x_t)$. To prove that, consider two cases: if $g_t(y_t|x_t) \geq c_t$, the result of the multiplication is $g_t(y_t|x_t) \times 1 = g_t(y_t|x_t)$; if $g_t(y_t|x_t) < c_t$, the result is $c_t \times g_t(y_t|x_t)/c_t = g_t(y_t|x_t)$. (This also gives an intuition about why the weight gets lifted to c_t in the case of acceptance with $w_t < c_t$.) Using this we have that:

$$\begin{aligned} \mathbb{E}[w_t] &= \int g_t(y_t|x_t) \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t \left(x_t \middle| x_{t-1}^{(n)} \right) dx_t = \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} \int f_t \left(x_t \middle| x_{t-1}^{(n)} \right) g_t(y_t|x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p \left(y_t \middle| x_{t-1}^{(n)} \right). \end{aligned}$$

The number of propagations P_t is a random variable distributed according to the negative binomial distribution with the number of successes $N + 1$ and the probability of success p_{A_t} :

$$P(P_t = D) = \binom{D-1}{(N+1)-1} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)}.$$

The expected value of $\sum_{n=1}^N w_t^{(n)} / (P_t - 1)$ is given by

$$\begin{aligned} \mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1} \right] &= \sum_{D=N+1}^{\infty} \frac{N \mathbb{E}[w_t]}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N \mathbb{E}[w_t] \sum_{D=N+1}^{\infty} \frac{1}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} = N \mathbb{E}[w_t] \frac{p_{A_t}}{N} \\ &= \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p \left(y_t \middle| x_{t-1}^{(n)} \right). \end{aligned}$$

□

The rest of the proof is identical to the proof in [14], which we include below for completeness.

Lemma 2.

$$\mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)} p(y_{t+1:t'} | x_t^{(n)})}{P_t - 1} \middle| \mathcal{S}_{t-1} \right] = \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}).$$

Proof. Similar to the proof of Lemma 1 we have that

$$\begin{aligned} \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] &= \int \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} f_t(x_t | x_{t-1}^{(n)}) g_t(y_t | x_t) p(y_{t+1:t'} | x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} \int f_t(x_t | x_{t-1}^{(n)}) g_t(y_t | x_t) p(y_{t+1:t'} | x_t) dx_t \\ &= \frac{1}{p_{A_t}} \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}). \end{aligned}$$

Using this result we have that

$$\begin{aligned} \mathbb{E} \left[\frac{\sum_{n=1}^N w_t^{(n)} p(y_{t+1:t'} | x_t^{(n)})}{P_t - 1} \right] &= \sum_{D=N+1}^{\infty} \frac{N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)]}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] \sum_{D=N+1}^{\infty} \frac{1}{D-1} \binom{D-1}{N} p_{A_t}^{N+1} (1-p_{A_t})^{D-(N+1)} \\ &= N \mathbb{E}[w_t p(y_{t+1:t'} | x_t)] \frac{p_{A_t}}{N} = \sum_{n=1}^N \frac{w_{t-1}^{(n)}}{\sum_{m=1}^N w_{t-1}^{(m)}} p(y_{t:t'} | x_{t-1}^{(n)}). \end{aligned}$$

□

Lemma 3.

$$\mathbb{E} \left[\prod_{t'=t-h}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{S}_{t-h-1} \right] = \sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{\sum_{m=1}^N w_{t-h-1}^{(m)}} p(y_{t-h:t} | x_{t-h-1}^{(n)}).$$

Proof. By induction. The base step for $h = 0$ was proved in Lemma 1. In the induction step, let us assume that the equality holds for h and prove it for $h + 1$:

$$\begin{aligned} \mathbb{E} \left[\prod_{t'=t-h-1}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{S}_{t-h-2} \right] &= \mathbb{E} \left[\mathbb{E} \left[\prod_{t'=t-h}^t \frac{\sum_{n=1}^N w_{t'}^{(n)}}{P_{t'} - 1} \middle| \mathcal{S}_{t-h-1} \right] \frac{\sum_{n=1}^N w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} \middle| \mathcal{S}_{t-h-2} \right] \\ &\quad \text{(using the induction assumption)} \\ &= \mathbb{E} \left[\sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{\sum_{m=1}^N w_{t-h-1}^{(m)}} p(y_{t-h:t} | x_{t-h-1}^{(n)}) \frac{\sum_{n=1}^N w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} \middle| \mathcal{S}_{t-h-2} \right] \\ &= \mathbb{E} \left[\sum_{n=1}^N \frac{w_{t-h-1}^{(n)}}{P_{t-h-1} - 1} p(y_{t-h:t} | x_{t-h-1}^{(n)}) \middle| \mathcal{S}_{t-h-2} \right] \\ &\quad \text{(using Lemma 2)} \\ &= \sum_{n=1}^N \frac{w_{t-h-2}^{(n)}}{\sum_{m=1}^N w_{t-h-2}^{(m)}} p(y_{t-h-1:t} | x_{t-h-2}^{(n)}). \end{aligned}$$

□

Theorem.

$$\mathbb{E} \left[\prod_{t=1}^T \frac{\sum_{n=1}^N w_t^{(n)}}{P_t - 1} \right] = p(y_{1:T}).$$

Proof. Using Lemma 3 with $t = T, h = T - 1$ and

$$\mathbb{E} \left[\frac{1}{N} \sum_{n=1}^N p(y_{1:T} | x_0^{(n)}) \right] = p(y_{1:T}).$$

□

B. BIASEDNESS OF THE MARGINAL LIKELIHOOD ESTIMATOR WITH DYNAMIC THRESHOLDS

Consider the following example: There are two coins on a table, one fair (F) and one biased (B). The probability of getting head (H) and tail (T) using the biased coin are 0.8 and 0.2, respectively. We choose a coin (uniformly at random) and flip it. The probability that the outcome will be head is

$$\begin{aligned} p(Y = H) &= p(Y = H | X = F)p(X = F) + p(Y = H | X = B)p(X = B) = \\ &= 0.5 \times 0.5 + 0.8 \times 0.5 = 0.65, \end{aligned}$$

where X denotes the selected coin (either F or B) and Y denotes the observed outcome (either H or T).

Although this is quite a trivial example, we can still use it to show that using dynamic thresholds, such as quantiles determined in each sweep, may lead to a biased estimate of the marginal likelihood. Note that if the thresholds are constant in all sweeps, the marginal likelihood is unbiased as shown in the proof in Appendix A.

Let us employ a particle filter with rejection control (although there is only one time step) to estimate $p(Y = H)$. We will use the median of the candidate weights after the first propagation of each particle (including the additional particle) as the threshold. To demonstrate the biasedness of the estimator, we consider a filter with only one particle (i.e., $N = 1$). The estimator of the marginal likelihood is in this case given by:

$$\hat{Z} = \frac{w^{(1)}}{P - 1}.$$

There exist four possible states of the initial candidate particles (including the additional particle) after the propagation step:

Case	$x^{(1)}$	$x^{(2)}$	$w^{(1)}$	$w^{(2)}$	Median
1	H	H	0.8	0.8	0.8
2	T	T	0.5	0.5	0.5
3	H	T	0.8	0.5	0.65
4	T	H	0.5	0.8	0.65

Each of these cases is equally likely (the probability of each one being 0.25).

In the first case, both weights are equal to the threshold so both particles are accepted, $w^{(1)} = 0.8, P = 2$ and $\mathbb{E}[\hat{Z} | \text{case 1}] = 0.8$.

Using a similar reasoning we get $\mathbb{E}[\hat{Z} | \text{case 2}] = 0.5$ for the second case.

The remaining cases are more difficult. Let $p_F = 0.5/0.65$ denote the acceptance probability of a particle with weight $w' = 0.5$, and $p_A = 0.5 \times 1 + 0.5 \times 0.5/0.65$ denote the acceptance probability of a restarted particle (its weight being irrelevant).

In the third case, the weight $w^{(1)}$ of the accepted particle is 0.8, but the number P of propagations varies. The expected value of \hat{Z} is given by

$$\begin{aligned} \mathbb{E}[\hat{Z} | \text{case 3}] &= p_F \frac{0.8}{2 - 1} + (1 - p_F) \sum_{P=3}^{\infty} \frac{0.8}{P - 1} (1 - p_A)^{P-3} p_A \\ &= 0.8 \left(p_F + (1 - p_F) \frac{p_A (p_A - \log(p_A) - 1)}{(1 - p_A)^2} \right) \approx 0.70392 \end{aligned}$$

The fourth case is even more complicated, as we need to distinguish between the case where the weight $w^{(1)}$ of the accepted particle is 0.8 and the case where the weight is 0.65:

$$\begin{aligned} \mathbb{E}[\widehat{Z}|\text{case 4}] &= p_F \frac{0.65}{2-1} + (1-p_F) \sum_{P=3}^{\infty} \frac{0.8}{P-1} (1-p_A)^{P-3} 0.5 + (1-p_F) \sum_{P=3}^{\infty} \frac{0.65}{P-1} (1-p_A)^{P-3} 0.5 \frac{0.5}{0.65} \\ &= 0.65 \left(p_F + (1-p_F) \frac{p_A - \log(p_A) - 1}{(1-p_A)^2} \right) \approx 0.58132 \end{aligned}$$

Finally, we can show that the expected value of \widehat{Z} is not equal to $p(Y = H)$:

$$\mathbb{E}[\widehat{Z}] = 0.25 \mathbb{E}[\widehat{Z}|\text{case 1}] + 0.25 \mathbb{E}[\widehat{Z}|\text{case 2}] + 0.25 \mathbb{E}[\widehat{Z}|\text{case 3}] + 0.25 \mathbb{E}[\widehat{Z}|\text{case 4}] \approx 0.64631 \neq 0.65.$$

