Delayed Sampling and Automatic Rao–Blackwellization of Probabilistic Programs

Lawrence Murray[1], Daniel Lundén[2], Jan Kudlicka[1], David Broman[2], Thomas Schön[1]
[1]Uppsala University and [2]KTH Royal Institute of Technology

1. Motivation

- Probabilistic programming languages often perform inference using the **bootstrap particle filter**.
- We would like to enable variance reduction techniques such as **Rao–Blackwellization**, **locally-optimal proposals**, and **variable elimination**.
- Ideally, this should be **automatic**, without changes to program code.

2. Idea

As they execute, probabilistic programs typically trigger checkpoints of two types:

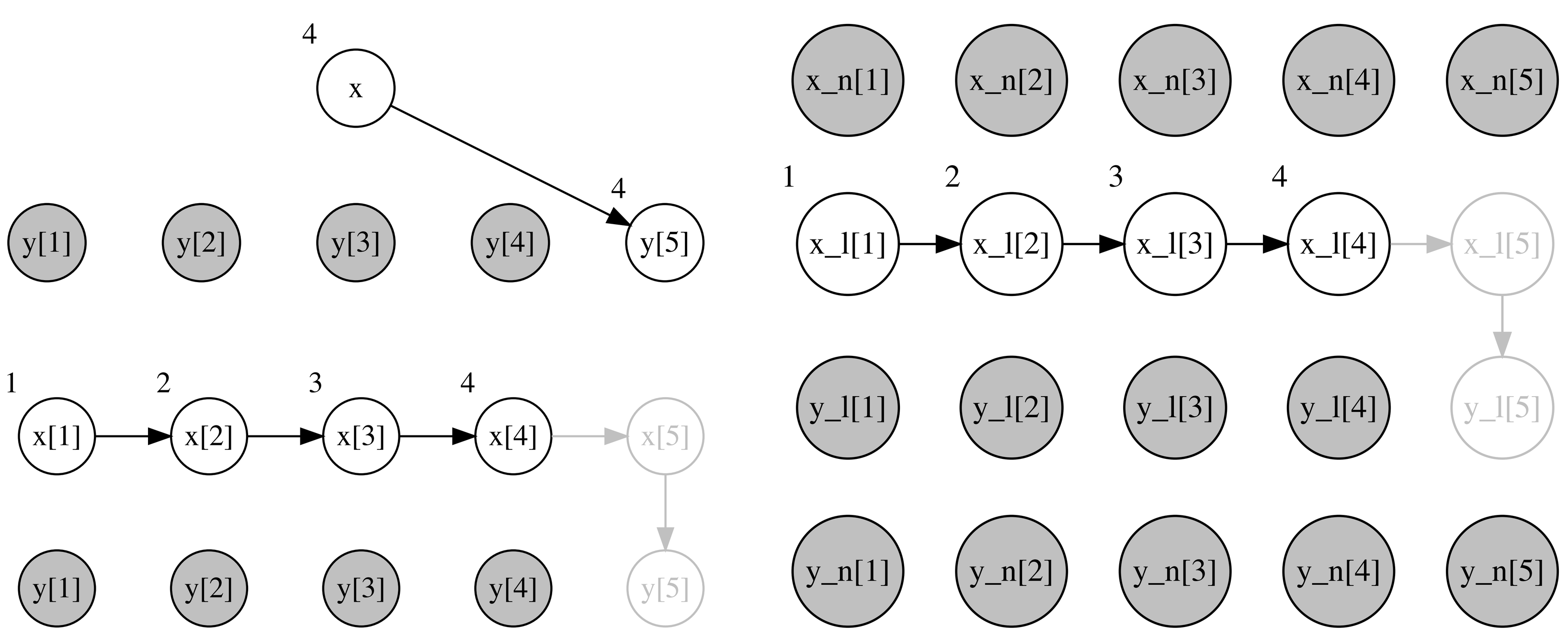
- **sample** to eagerly sample a random variable, and
- **observe** to update a weight given some value for a random variable.

We instead use three types:

- **assume** to initialize a random variable with some distribution,
- **value** to instantiate such a random variable, and
- **observe** to condition given some value for a random variable.

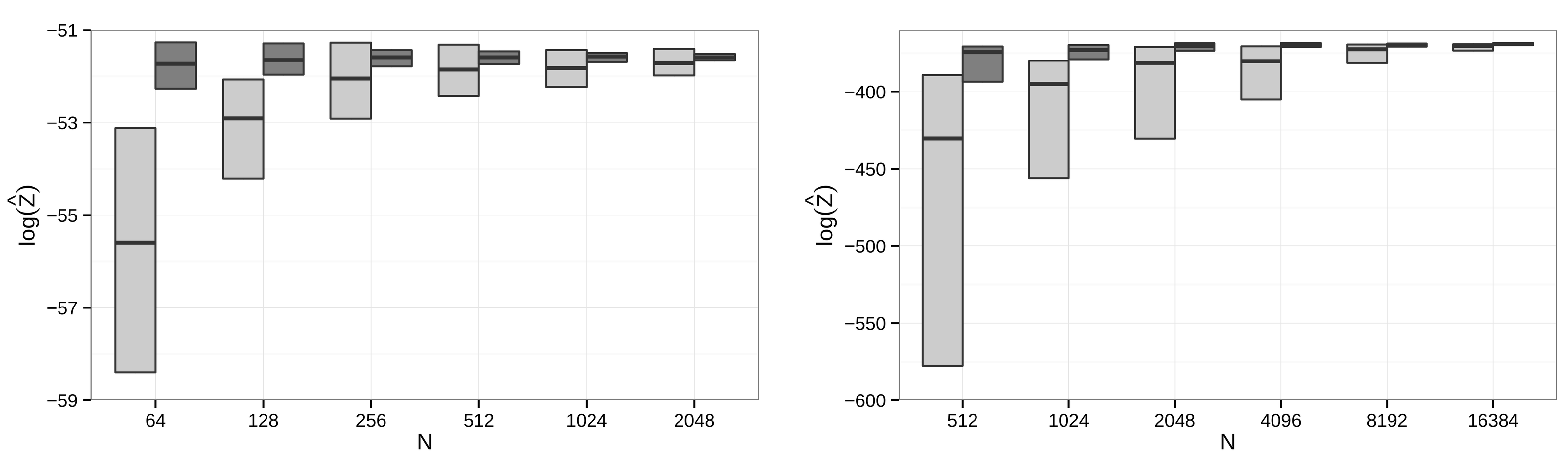
These three types facilitate **delayed sampling**. Between **assume** and **value** checkpoints, the distribution of a random variable can be updated at **observe** checkpoints, using analytical relationships such as conjugate priors and affine transformations.

The analytical relationships are maintained in a directed graph alongside the running program. Checkpoints trigger operations on this graph, such as insertion, marginalization, observation and sampling.



3. Benefits

This can significantly reduce variance in marginal likelihood estimates (left, dark gray) versus a bootstrap particle filter (right, light gray).



For a linear-nonlinear state-space model, delayed sampling marginalizes out the linear component of the state to automatically produce a Rao–Blackwellized particle filter.

Similarly, for an epidemiological model, delayed sampling marginalises out the parameters, producing a random-weight or pseudo-marginal-style importance sampler with similar improvements.

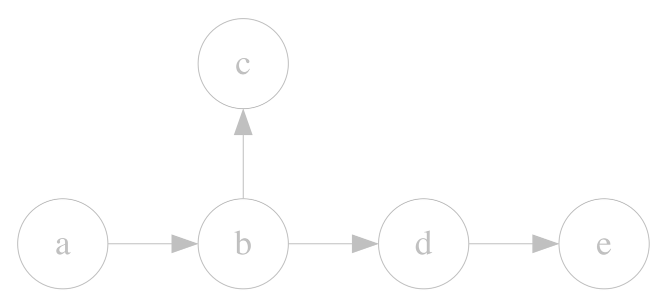
4. Implementation

Delayed sampling has been implemented in Anglican and Birch, a new universal probabilistic programming language.

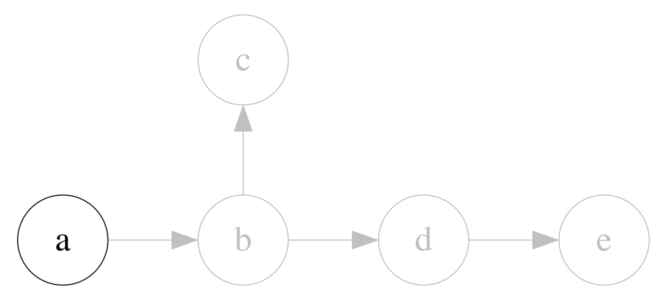
5. Worked Example

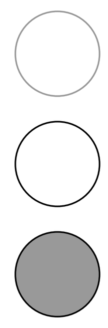
Code	Checkpoint
`a ~ Gaussian(0.0, 1.0);`	assume(a)
`b ~ Gaussian(a, 1.0);`	assume(b)
`c ~ Gaussian(b, 1.0);`	assume(c)
`d ~ Gaussian(b, 1.0);`	assume(d)
`e ~ Gaussian(d, 1.0);`	observe(e)
`print(c);`	value(c)

1.

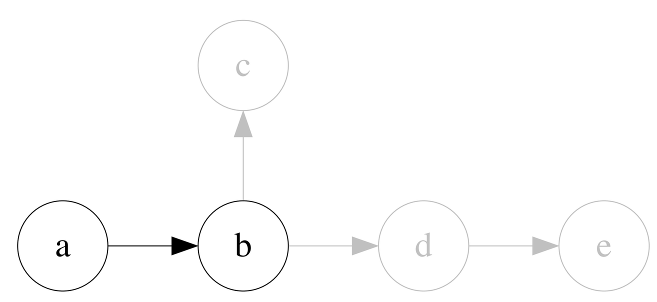


2.



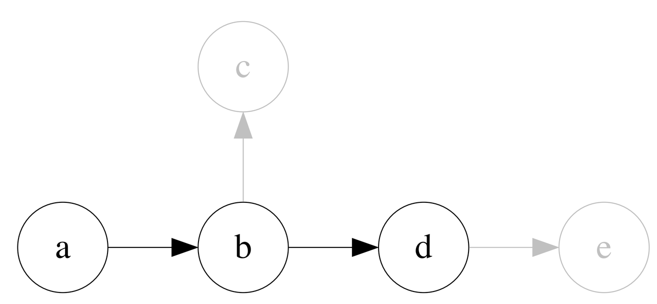


3.

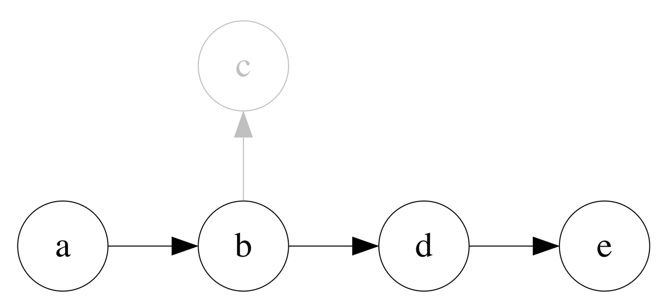


A number annotating a node indicates the number of observations on which it has been conditioned.

4.

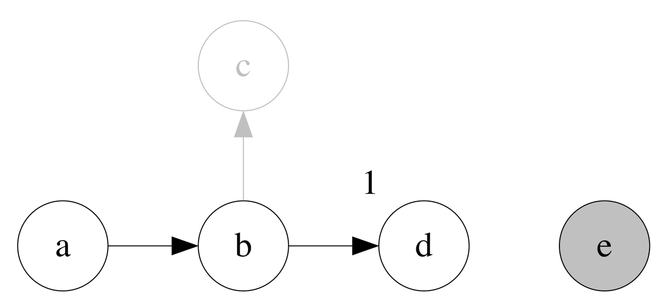


5.



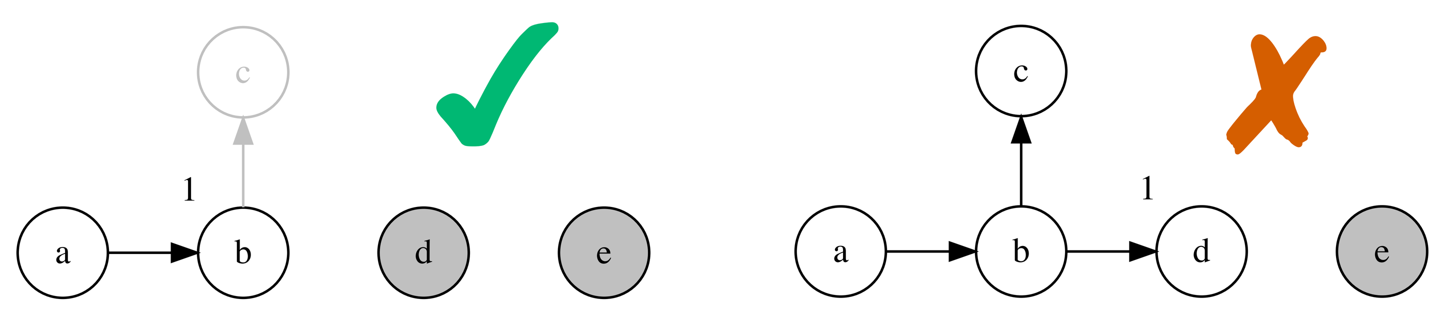
Marginalized nodes must form a single path, called the **M-path**, extending from the root.

6.

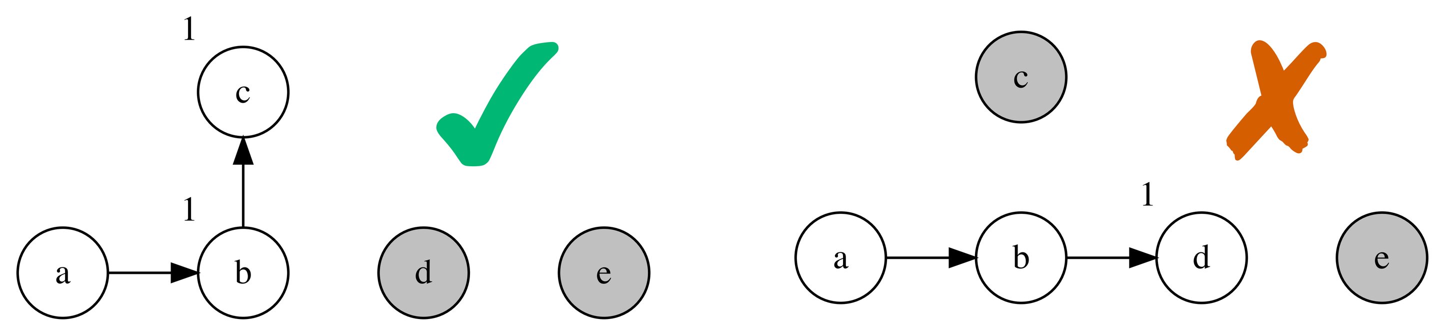


Below, this rule is violated, and sampling of c does not benefit from the observation of e.

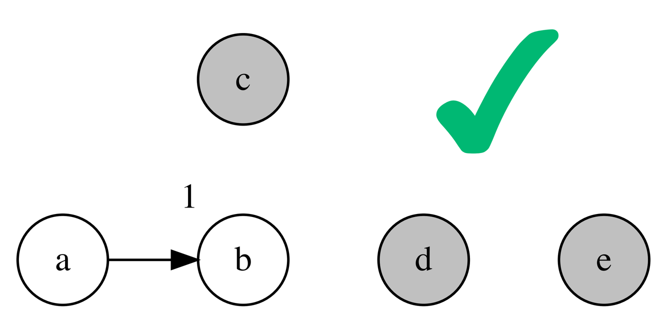
7.



8.



9.



The fix is to retract the **M-path** before extending it to a node to be sampled or observed.

www.birch-lang.org

UPPSALA UNIVERSITET

KTH VETENSKAP OCH KONST

Swedish Foundation for Strategic Research