Output-Sensitive Adaptive Metropolis-Hastings for Probabilistic Programs

David Tolpin(✉), Jan-Willem van de Meent, Brooks Paige, and Frank Wood

Department of Engineering Science, University of Oxford, Oxford, England
{dtolpin,jwvdm,brooks,fwood}@robots.ox.ac.uk

Abstract. We introduce an adaptive output-sensitive Metropolis-Hastings algorithm for probabilistic models expressed as programs, Adaptive Lightweight Metropolis-Hastings (AdLMH). This algorithm extends Lightweight Metropolis-Hastings (LMH) by adjusting the probabilities of proposing random variables for modification to improve convergence of the program output. We show that AdLMH converges to the correct equilibrium distribution and compare convergence of AdLMH to that of LMH on several test problems to highlight different aspects of the adaptation scheme. We observe consistent improvement in convergence on the test problems.

Keywords: Probabilistic programming · Adaptive MCMC

1 Introduction

One strategy for improving convergence of Markov Chain Monte Carlo (MCMC) samplers is through online adaptation of the proposal distribution [1,2,15]. An adaptation scheme must ensure that the sample sequence converges to the correct equilibrium distribution. In a componentwise updating Metropolis-Hastings MCMC sampler, i.e. Metropolis-within-Gibbs [5,8,10], the proposal distribution can be decomposed into two components:

1. A stochastic schedule (probability distribution) for selecting the next random variable for modification.
2. The kernels from which new values for each of the variables are proposed.

In this paper we concentrate on the first component—adapting the schedule for selecting a variable for modification. Our primary interest in this work is to improve MCMC methods for probabilistic programming [6,7,11,13,17]. Probabilistic programming languages facilitate development of probabilistic models using the expressive power of general programming languages. The goal of inference in such programs is to reason about the posterior distribution over random variates that are sampled during execution, conditioned on observed values that constrain a subset of program expressions.

Lightweight Metropolis-Hastings (LMH) samplers [16] propose a change to a single random variable at each iteration. The program is then rerun, reusing

A. Appice et al. (Eds.): ECML PKDD 2015, Part II, LNAI 9285, pp. 311–326, 2015.
DOI: 10.1007/978-3-319-23525-7_19

previous values and computation where possible, after which the new set of sample values is accepted or rejected. While re-running the program each time may waste some computation, the simplicity of LMH makes developing probabilistic variants of arbitrary languages relatively straightforward.

Designing robust adaptive MCMC methods for probabilistic programming is complicated because of diversity of models that can be expressed as probabilistic programs. Ideally, a single adaptation scheme should perform well in different programs without requiring manual tuning of parameters. Here we present an adaptive variant of LMH that dynamically adjusts the schedule for selecting variables for modification. First, we review the general structure of a probabilistic program. We discuss convergence criteria with respect to the program output and propose a scheme for tracking the "influence" of each random variable on the output. We then adapt the selection probability for each variable, borrowing techniques from the upper confidence bound (UCB) family of algorithms for multi-armed bandits [3]. We show that the proposed adaptation scheme preserves convergence to the target distribution under reasonable assumptions. Finally, we compare original and Adaptive LMH on several test problems to show how convergence is improved by adaptation.

2 Preliminaries

2.1 Probabilistic Program

A probabilistic program is a stateful deterministic computation $\mathcal{P}$ with the following properties:

- Initially, $\mathcal{P}$ expects no arguments.
- On every call, $\mathcal{P}$ returns either a distribution and an address (F, α), a distribution and a value (G, y), a value z, or $\perp$.
- Upon returning F, $\mathcal{P}$ expects a value x drawn from F as the argument to the next call.
- Upon returning (G, y) or z, $\mathcal{P}$ is invoked again without arguments.
- Upon returning $\perp$, $\mathcal{P}$ terminates.

A program is run by calling $\mathcal{P}$ repeatedly until termination.

A program need not generate the same sequence of random variables in every execution. For this reason we assume that each random variable x is assigned a unique label α, which we call an address, that induces a correspondence between variables in different executions. Every execution implicitly produces a sequence of triples of distributions, values of *latent* random variables, and addresses, (F_i, x_i, α_i). We call this sequence a *trace* and denote it by $\boldsymbol{x}$. A trace induces a sequence of pairs (G_j, y_j) of distributions and values of *observed* random variables. We call this sequence an *image* and denote it by $\boldsymbol{y}$. For notational simplicity we assume that the program always generates the same ordered set $\boldsymbol{z}$ of *output* values z_k.

The target density $\pi(\boldsymbol{x}) := \gamma(\boldsymbol{x})/Z$ of a program is defined in terms of the product of the probabilities of all random choices $\boldsymbol{x}$ and the likelihood of all observations $\boldsymbol{y}$

$$\gamma(\boldsymbol{x}) := \prod_{i=1}^{|\boldsymbol{x}|} p_{F_i}(x_i) \prod_{j=1}^{|\boldsymbol{y}|} p_{G_j}(y_j). \tag{1}$$

The objective of inference in probabilistic program $\mathcal{P}$ is to discover the distribution of $\boldsymbol{z}$.

2.2 Adaptive Markov Chain Monte Carlo

MCMC methods generate a sequence of samples $\{\boldsymbol{x}^t\}_{t=1}^{\infty}$ by simulating a Markov chain using a transition operator that leaves a target density $\pi(\boldsymbol{x})$ invariant. In MH the transition operator is implemented by drawing a new sample $\boldsymbol{x}'$ from a parameterized proposal distribution $q_\theta(\boldsymbol{x}'|\boldsymbol{x}^t)$ that is conditioned on the current sample $\boldsymbol{x}^t$. The proposed sample is then accepted with probability

$$\rho = \min\left(\frac{\pi(\boldsymbol{x}')q_\theta(\boldsymbol{x}^t|\boldsymbol{x}')}{\pi(\boldsymbol{x}^t)q_\theta(\boldsymbol{x}'|\boldsymbol{x}^t)}, 1\right). \tag{2}$$

If $\boldsymbol{x}'$ is rejected, $\boldsymbol{x}^t$ is re-used as the next sample.

The convergence rate of MH depends on parameters θ of the proposal distribution q_θ. The parameters can be set either offline or online. Variants of MCMC in which the parameters are continuously adjusted based on the features of the sample sequence are called adaptive. Challenges in design and analysis of Adaptive MCMC methods include optimization criteria and algorithms for the parameter adaptation, as well as conditions of convergence of adaptive MCMC to the correct equilibrium distribution [14]. Continuous adaptation of parameters of the proposal distribution is a well-known research subject [1,2,15].

In a componentwise MH algorithm [10] that targets a density $\pi(\boldsymbol{x})$ defined on an N-dimensional space $\mathcal{X}$, the components of a sample $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ are updated individually, in either random or systematic order. Assuming the component i is selected at the step t for modification, the proposal $\boldsymbol{x}'$ sampled from $q_\theta^i(\boldsymbol{x}|\boldsymbol{x}^t)$ may differ from $\boldsymbol{x}^t$ only in that component, and $x'_j = x_j^t$ for all $j \neq i$. Adaptive componentwise Metropolis-Hastings (Algorithm 1) chooses different probabilities for selecting a component for modification at each iteration. Parameters of this scheduling distribution may be viewed as a subset of parameters θ of the proposal distribution q_θ, and adjusted according to optimization criteria of the sampling algorithm.

Varying selection probabilities based on past samples violates the Markov property of $\{\boldsymbol{x}^t\}_1^{\infty}$. However, provided the change in selection probabilities decreases to zero as t approaches ∞, then under suitable regularity conditions for the target density (see Section 4) an adaptive componentwise MH algorithm will be ergodic [8], and the distribution on $\boldsymbol{x}$ induced by Algorithm 1 converges to π.

Algorithm 1. Adaptive componentwise MH

1: Select initial point $\boldsymbol{x}^0$.
2: Set initial selection probabilities $\boldsymbol{w}^0$.
3: **for** $t = 1 \ldots \infty$ **do**
4: $\quad \boldsymbol{w}^t \leftarrow f^t(\boldsymbol{w}^{t-1}, \boldsymbol{x}^0, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^t)$.
5: $\quad$ Choose $k \in \{1, \ldots, N\}$ with probability w_k^t.
6: $\quad$ Generate $\boldsymbol{x}' \sim q_\theta^k(\boldsymbol{x}|\boldsymbol{x}^t)$.
7: $\quad \rho \leftarrow \min\left(\frac{\pi(\boldsymbol{x}')q_\theta^k(\boldsymbol{x}^t|\boldsymbol{x}')}{\pi(\boldsymbol{x}^t)q_\theta^k(\boldsymbol{x}'|\boldsymbol{x}^t)}, 1\right)$
8: $\quad \boldsymbol{x}^{t+1} \leftarrow \boldsymbol{x}'$ with probability ρ, $\boldsymbol{x}^t$ otherwise.
9: **end for**

2.3 Lightweight Metropolis-Hastings

LMH [16] is a sampling scheme for probabilistic programs where a single random variable drawn in the course of a particular execution of a probabilistic program is modified via a standard MH proposal. LMH differs from componentwise MH algorithms in that other random variables may also have to be modified, depending on the structural dependencies in the probabilistic program.

LMH initializes a proposal by selecting a single variable x_k at address α_k from an execution trace $\boldsymbol{x}$ and resampling its value x'_k either using a reversible kernel $\kappa(x'_k|x_k)$ or from the conditional prior F_k. The remainder of the program is then rerun to generate a new trace $\boldsymbol{x}'$. When generating a variable x'_j at address α'_j in the new trace, the value x_i from the previous trace such that $\alpha'_j = \alpha_i$ is reused, provided it exists and still lies in the support of F'_j. When no value can be rescored, a new value x'_j is sampled from F'_j. The acceptance probability ρ_{LMH} is obtained by substituting (1) into (2):

$$\rho_{\text{LMH}} = \min\left(1, \frac{p(\boldsymbol{y}'|\boldsymbol{x}')p(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})}\right). \tag{3}$$

We here further simplify LMH by assuming x'_i is sampled from the conditional prior F_i and that all variables are selected for modification with equal probability. Under these assumptions, ρ_{LMH} takes the form [17]

$$\rho_{\text{LMH}} = \min\left(1, \frac{p(\boldsymbol{y}'|\boldsymbol{x}')p(\boldsymbol{x}')|\boldsymbol{x}|p(\boldsymbol{x}\setminus\boldsymbol{x}'|\boldsymbol{x}\cap\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})|\boldsymbol{x}'|p(\boldsymbol{x}'\setminus\boldsymbol{x}|\boldsymbol{x}'\cap\boldsymbol{x})}\right), \tag{4}$$

where $\boldsymbol{x}'\setminus\boldsymbol{x}$ denotes the resampled variables, and $\boldsymbol{x}'\cap\boldsymbol{x}$ denotes the variables which have the same values in both traces.

3 Adaptive Lightweight Metropolis-Hastings

We develop an adaptive variant of LMH that dynamically adjusts the probabilities of selecting variables for modification (Algorithm 2). Let $\boldsymbol{x}^t$ be the trace at iteration t of Adaptive LMH. We define the probability distribution of selecting variables for modification in terms of an indexed set of weights $\boldsymbol{W}^t$ that we

Algorithm 2. Adaptive LMH

```
Initialize W^0_α to a constant for all addresses α.
Run the program.
for t = 1 ... ∞ do
    Randomly select a variable x^t_k according to W^t.
    Propose a value for x^t_k.
    Run the program, accept or reject the trace.
    if accepted then
        Compute W^{t+1} based on the program output.
    else
        W^{t+1} ← W^t
    end if
end for
```

adapt, such that the probability w_k^t of selecting the variable at address α_k for modification is

$$w_k^t := W_{\alpha_k}^t \Big/ \sum_{i=1}^{|\boldsymbol{x}^t|} W_{\alpha_i}^t. \tag{5}$$

Just like LMH, Adaptive LMH runs the probabilistic program once and then selects variables for modification randomly. However, the acceptance ratio ρ_{AdLMH} must now include selection probabilities w_k and w'_k of the resampled variable in the current and the proposed sample

$$\rho_{\text{AdLMH}} = \min\left(1, \frac{p(\boldsymbol{y}'|\boldsymbol{x}')p(\boldsymbol{x}')w'_k p(\boldsymbol{x}\backslash\boldsymbol{x}'|\boldsymbol{x}\cap\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})w_k p(\boldsymbol{x}'\backslash\boldsymbol{x}|\boldsymbol{x}'\cap\boldsymbol{x})}\right). \tag{6}$$

This high-level description does not detail how $\boldsymbol{W}^t$ is computed for each iteration. Indeed, this is the most essential part of the algorithm. There are two different aspects here — on one hand, the influence of a given choice on the output sequence must be quantified in terms of convergence of the sequence to the target distribution. On the other hand, the influence of the choice must be translated into re-computation of weights of random variables in the trace. Both parts of re-computation of $\boldsymbol{W}^t$ are explained below.

3.1 Quantifying Influence

Extensive research has been devoted to criteria for tuning parameters of adaptive MCMC [1,2,15]. The case of inference in probabilistic programs is different: the user of a probabilistic program is often interested in fast convergence of the program output $\{\boldsymbol{z}^t\}$ rather than the trace $\{\boldsymbol{x}^t\}$.

In adaptive MCMC variants the acceptance rate can be efficiently used as the optimization objective [15]. However, for convergence of the output sequence an accepted trace that produces the same output is indistinguishable from a rejected trace. Additionally, while optimal values of the acceptance rate [1,15] can be used to tune parameters in adaptive MCMC, in Adaptive LMH we do

not change the parameters of proposal distributions of individual variables, and assume that they are fixed. However, proposing a new value for a random variable may or may not change the output even if the new trace is accepted. By changing variable selection probabilities we attempt to maximize the change in the output sequence so that it converges faster. In the pedagogical example

$$x_1 \sim \text{Bernoulli}(0.5), \quad x_2 \sim \mathcal{N}(x_1, 1),$$
$$z_1 \leftarrow (x_1, x_2),$$

selecting the Bernoulli random choice for modification changes the output only when a different value is sampled, while selecting the normal random choice will change the output almost always.

Based on these considerations, we quantify the influence of sampling on the output sequence by measuring the change in the output $\boldsymbol{z}$ of the probabilistic program. Since programs may produce output of any type, we chose to discern between identical and different outputs only, rather than to quantify the distance by introducing a type-dependent norm. In addition, when $|\boldsymbol{z}| > 1$, we quantify the difference by the fraction of components of $\boldsymbol{z}$ with changed values.

Formally, let $\{\boldsymbol{z}^t\}_1^\infty = \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^{t-1}, \boldsymbol{z}^t, \ldots\}$ be the output sequence of a probabilistic program. Then the *influence* of a choice that produced $\boldsymbol{z}^t$ is defined by the total reward R^t, computed as normalized Hamming distance

$$R^t = \frac{1}{|\boldsymbol{z}^t|} \sum_{l=1}^{|\boldsymbol{z}^t|} \mathbf{1}(z_l^t \neq z_l^{t-1}). \tag{7}$$

The reward is used to adjust the variable selection probabilities for the subsequent steps of Adaptive LMH by computing $\boldsymbol{W}^{t+1}$ (line 8 of Algorithm 2). It may seem sufficient to assign the reward to the last choice and use average choice rewards as their weights, but this approach will not work for Adaptive LMH. Consider the generative model

$$x_1 \sim \mathcal{N}(1, 10), \quad x_2 \sim \mathcal{N}(x_1, 1),$$
$$y_1 \sim \mathcal{N}(x_2, 1),$$
$$z_1 \leftarrow x_1,$$

where we observe the value $y_1 = 2$. Modifying x_2 may result in an accepted trace, but the value of $z_1 = x_1$, predicted by the program, will remain the same as in the previous trace. Only when x_1 is also modified, and a new trace with the updated values for both x_1 and x_2 is accepted, the earlier change in x_2 is indirectly reflected in the output of the program. In the next section, we discuss propagation of rewards to variable selection probabilities in detail.

3.2 Propagating Rewards to Variables

Both LMH and Adaptive LMH modify a single variable per trace, and either re-use or recompute the probabilities of values of all other variables

Algorithm 3. Propagating Rewards to Variables

```
1: for l in 1, ..., |z^t| do
2:     Append α to history h_l
3:     if z_l^{t+1} ≠ z_l^t then
4:         δ ← 1/|h_l|
5:         for α' in h_l do
6:             r_α' ← r_α' + δ, c_α' ← c_α' + δ
7:         end for
8:         Flush h_l.
9:     else
10:        c_α ← c_α + 1
11:    end if
12: end for
```

(except those absent from the previous trace or having an incompatible distribution, for which new values are also sampled). Due to this updating scheme, the influence of modifying a variable on the output can be delayed by several iterations. We propose the following propagation scheme: for each unique random variable x at address α, the reward r_α and count c_α are kept in a data structure used to compute $\boldsymbol{W}$. A list of addresses selected for modification since the last change in output, which we call the history $\boldsymbol{h}_l$, is maintained for each component z_l of the output $\boldsymbol{z}$. When the value of z_l changes, the reward is distributed between all of the addresses in the history $\boldsymbol{h}_l$, which is then emptied. When z_l does not change, the selected variable is penalized by zero reward. This scheme is shown in Algorithm 3 which expands line 8 of Algorithm 2.

Rewarding all of the variables in the history ensures that while variables which cause changes in the output more often get a greater reward, variables with lower influence are still selected for modification sufficiently often. This, in turn, ensures ergodicity of sampling sequence, and helps establish conditions for convergence to the target distribution, as we discuss in Section 4.

Let us show that under certain assumptions the proposed reward propagation scheme has a non-degenerate equilibrium for variable selection probabilities. Indeed, assume that for a program with two variables, x_1, and x_2, *probability matching*, or selecting a choice with the probability proportional to the unit reward $\rho_i = \frac{r_i}{c_i}$, is used to compute the weights, that is, $W_i = \rho_i$. Then, the following lemma holds:

Lemma 1. *Assume that for variables x_i, where $i \in \{1, 2\}$:*

- *w_i is the selection probability;*
- *β_i is the probability that the new trace is accepted given that the variable was selected for modification;*
- *γ_i is the probability that the output changed given that the trace was accepted.*

Assume further that w_i, β_i, and γ_i are constant. Then $\gamma_1 = 1$, $\gamma_2 = 0$ implies:

$$0 < \frac{w_2}{w_1} \leq \frac{1}{3} \tag{8}$$

Proof. We shall prove the lemma in three steps. First, we will analyze a sequence of samples between two subsequent arrivals of x_1. Then, we derive a formula for the expected unit reward of x_2. Finally, we shall bound the ratio $\frac{w_2}{w_1}$.

Consider a sequence of k samples, for some k, between two subsequent arrivals of x_1, including the sample corresponding to the second arrival of x_1. Since a new value of x_1 always ($\gamma_1 = 1$) and x_2 never ($\gamma_2 = 0$) causes a change in the output, at the end of the sequence the history will contain k occurrences of x_2. Let us denote by Δr_i, Δc_i the increase of reward r_i and count c_i between the beginning and the end of the sequence. Noting that x_2 is penalized each time it is added to the history (line 10 of Algorithm 3), and k occurrences of x_2 are rewarded when x_1 is added to the history (line 6 of Algorithm 3), we obtain

$$\Delta r_1 = \frac{1}{k+1},\ \Delta c_1 = \frac{1}{k+1} \quad \Delta r_2 = \frac{k}{k+1},\ \Delta c_2 = k + \frac{k}{k+1} \tag{9}$$

Consider now a sequence of M such sequences. When $M \to \infty$, $\frac{r_{iM}}{c_{iM}}$ approaches the expected unit reward $\overline{\rho}_i$, where r_{iM} and c_{iM} are the reward and the count of x_i at the end of the sequence.

$$\overline{\rho}_i = \lim_{M\to\infty} \frac{r_{iM}}{c_{iM}} = \lim_{M\to\infty} \frac{\frac{r_{iM}}{M}}{\frac{c_{iM}}{M}} = \lim_{M\to\infty} \frac{\frac{\sum_{m=1}^{M} \Delta r_{im}}{M}}{\frac{\sum_{m=1}^{M} \Delta c_{im}}{M}} = \frac{\overline{\Delta r_i}}{\overline{\Delta c_i}} \tag{10}$$

Each variable x_i is selected randomly and independently and produces an accepted trace with probability

$$p_i = \frac{w_i \beta_i}{w_1 \beta_1 + w_2 \beta_2}. \tag{11}$$

Acceptances of x_1 form a Poisson process with rate $\frac{1}{p_1} = \frac{w_1\beta_1 + w_2\beta_2}{w_1\beta_1}$, so k is geometrically distributed $\Pr[k] = (1-p_1)^k p_1$. Since $\Delta r_1 = \Delta c_1$ for any k, the expected unit reward $\overline{\rho}_1$ of x_1 is 1. We substitute $\overline{\Delta r_i}$ and $\overline{\Delta c_i}$ into (10) to obtain the expected unit reward $\overline{\rho}_2$ of x_2:

$$\begin{aligned}
\overline{\Delta r_2} &= \sum_{k=0}^{\infty} \frac{k}{k+1} (1-p_1)^k p_1 \\
\overline{\Delta c_2} &= \sum_{k=0}^{\infty} \left(k + \frac{k}{k+1}\right) (1-p_1)^k p_1 = \underbrace{\frac{1-p_1}{p_1}}_{\overline{k}} + \sum_{k=0}^{\infty} \frac{k}{k+1} (1-p_1)^k p_1
\end{aligned} \tag{12}$$

$$\overline{\rho}_2 = \frac{\overline{\Delta r_2}}{\overline{\Delta c_2}} = \frac{\sum_{k=0}^{\infty} \frac{k}{k+1}(1-p_1)^k p_1}{\frac{1-p_1}{p_1} + \sum_{k=0}^{\infty} \frac{k}{k+1}(1-p_1)^k p_1} = \frac{1 - \overbrace{\sum_{k=0}^{\infty} \frac{1}{k+1}(1-p_1)^k p_1}^{A}}{\frac{1}{p_1} - \underbrace{\sum_{k=0}^{\infty} \frac{1}{k+1}(1-p_1)^k p_1}_{A}} \tag{13}$$

For probability matching, selection probabilities are proportional to expected unit rewards:

$$\frac{w_2}{w_1} = \frac{\overline{\rho}_2}{\overline{\rho}_1} \tag{14}$$

To prove the inequality, we shall derive a closed-form representation for $\overline{\rho}_2$, and analyse solutions of (14) for $\frac{w_2}{w_1}$. We shall eliminate the summation A in (13):

$$\begin{aligned} A &= \sum_{k=0}^{\infty} \frac{1}{k+1}(1-p_1)^k p_1 = \frac{p_1}{1-p_1} \sum_{k=0}^{\infty} \frac{1}{k+1}(1-p_1)^{k+1} \\ &= \frac{p_1}{1-p_1} \sum_{k=0}^{\infty} \int_{p_1}^{1} (1-\xi)^k d\xi = \frac{p_1}{1-p_1} \int_{p_1}^{1} \sum_{k=0}^{\infty} (1-\xi)^k d\xi = -\frac{p_1}{1-p_1} \log p_1 \end{aligned} \tag{15}$$

By substituting A into (13), and then $\overline{\rho}_1$ and $\overline{\rho}_2$ into (14), we obtain

$$\frac{w_2}{w_1} = \frac{\overline{\rho}_2}{\overline{\rho}_1} = \overline{\rho}_2 = \left.\frac{1 + \frac{p_1 \log p_1}{1-p_1}}{\frac{1}{p_1} + \frac{p_1 \log p_1}{1-p_1}}\right\} B(p_1) \tag{16}$$

The right-hand side $B(p_1)$ of (16) is a monotonic function for $p_1 \in [0,1]$, and $B(0) = 0$, $B(1) = \frac{1}{3}$. According to (11), $\frac{w_2}{w_1} = 0$ implies $p_1 = 1$, hence $\frac{w_2}{w_1} \neq 0$, and $0 < \frac{w_2}{w_1} \leq \frac{1}{3}$. □

By noting that any subset of variables in a probabilistic program can be considered a single random variable drawn from a multi-dimensional distribution, Lemma 1 is generalized to any number of variables by Corollary 1:

Corollary 1. *For any partitioning of the set $\boldsymbol{x}$ of random variables of a probabilistic program, AdLMH with weights proportional to expected unit rewards selects variables from each of the partitions with non-zero probability.*

To ensure convergence of $\boldsymbol{W}^t$ to expected unit rewards in the stationary distribution, we use upper confidence bounds on unit rewards to compute the selection probabilities, an idea that we borrow from the UCB family of algorithms for multi-armed bandits [3]. Following UCB1 [3], we compute the upper confidence bound $\hat{\rho}_i$ as the sum of the unit reward and the exploration term

$$\hat{\rho}_\alpha = \rho_\alpha + C\sqrt{\frac{\log \sum_\alpha c_\alpha}{c_\alpha}}, \tag{17}$$

where C is an exploration factor. The default value for C is $\sqrt{2}$ in UCB1; in practice, a lower value of C is preferable. Note that variable selection in Adaptive LMH is different from arm selection in multi-armed bandits: unlike in bandits, where we want to sample the best arm at an increasing rate, in Adaptive LMH we expect $\boldsymbol{W}^t$ to converge to an equilibrium in which selection probabilities are proportional to expected unit rewards.

4 Convergence of Adaptive LMH

As adaptive MCMC algorithms may depend arbitrarily on the history at each step, showing that a given sampler correctly draws from the target distribution can be non-trivial. General conditions under which adaptive MCMC schemes are still ergodic, in the sense that the distribution of samples converges to the target π in total variation, are established in [14]. The fundamental criteria for validity of an adaptive algorithm are *diminishing adaptation*, which (informally) requires that the amount which the transition operator changes each iteration must asymptotically decrease to zero; and *containment*, a technical condition which requires that the time until convergence to the target distribution must be bounded in probability [4].

The class of models representable by probabilistic programs is very broad, allowing specification of completely arbitrary target densities; however, for many models the adaptive LMH algorithm reduces to an adaptive random scan Metropolis-within-Gibbs in Algorithm 1. To discuss when this is the case, we invoke the concept of *structural* versus *structure-preserving* random choices [18]. Crucially, a *structure-preserving* random choice x_k does not affect the existence of other x_m in the trace.

Suppose we were to restrict the expressiveness of our language to admit only programs with no structural random choices: in such a language, the LMH algorithm in Algorithm 2 reduces to the adaptive componentwise MH algorithm. Conditions under which such an adaptive algorithm is ergodic have been established explicitly in [8, Theorems4.10and5.5]. Given suitable assumptions on the target density defined by the program, it is necessary for the probability vector $||w^t - w^{t-1}|| \to 0$, and that for any particular component k we have probability $w_k^t > \epsilon > 0$. Both of these are satisfied by our approach: from Corollary 1, we ensure that the unit reward across each x_i converges to a positive fixed point.

While any theoretical result will require language restrictions such that programs only induce distributions satisfying regularity conditions, we conjecture that this scheme is broadly applicable across most non-pathological programs. We leave a precise theoretical analysis of the space of probabilistic programs in which adaptive MCMC schemes (with infinite adaptation) may be ergodic to future work. Empirical evaluation presented in the next section demonstrates practical convergence of Adaptive LMH on a range of inference examples, including programs containing structural random choices.

5 Empirical Evaluation

We evaluated Adaptive LMH on many probabilistic programs and observed consistent improvement of convergence rate compared to LMH. We also verified on a number of tests that the algorithm converges to the correct distribution obtained by independent exact methods. In this section, we compare Adaptive LMH to LMH on several representative examples of probabilistic programs. The rates in the comparisons are presented with respect to the number of samples, or

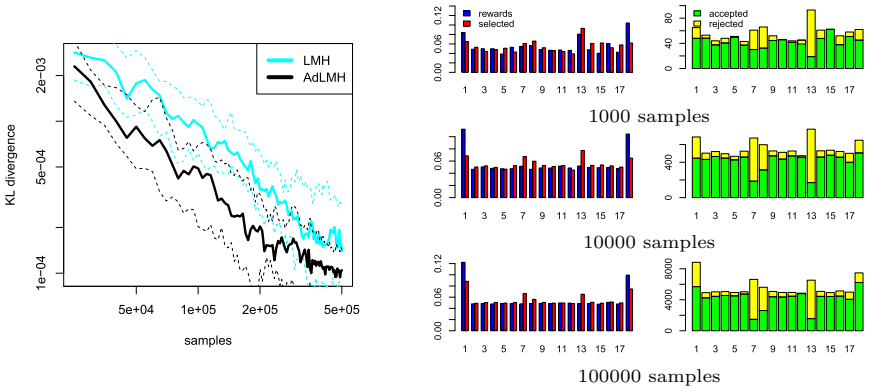

Fig. 1. HMM, predicting the 0th and 17th state

simulations, of the probabilistic programs. The additional computation required for adaptation takes negligible time, and the computational effort per sample is approximately the same for all algorithms. Our implementation of the inference engine is available at https://bitbucket.org/dtolpin/anglican.

In the following case studies differences between program outputs and target distributions are presented using Kullback-Leibler (KL) divergence, Kolmogorov-Smirnov (KS) distance, or L2 distance, as appropriate. In cases where target distributions cannot be updated exactly, they were approximated by running a non-adaptive inference algorithm for a long enough time and with a sufficient number of restarts. In each of the evaluations, all of the algorithms were run with 25 random restarts and 500 000 simulations of the probabilistic program per restart. The difference plots use the logarithmic scale for both axes. In the plots, the solid lines correspond to the median, and the dashed lines to 25% and 75% percentiles, taken over all runs of the corresponding inference algorithm. The exploration factor for computing upper confidence bounds on unit rewards (Equation 17) was fixed at $C = 0.5$ for all tests and evaluations.

The first example is a latent state inference problem in an HMM with three states, one-dimensional normal observations (0.9, 0.8, 0.7, 0, -0.025, 5, 2, 0.1, 0, 0.13, 0.45, 6, 0.2, 0.3, -1, -1) with variance 1.0, a known transition matrix, and known initial state distribution. There are 18 distinct random choices in all traces of the program, and the 0th and the 17th state are predicted. The results of evaluation are shown in Figure 1 as KL divergences between the inference output and the ground truth obtained using the forward-backward algorithm. In addition, bar plots of unit reward and sample count distributions among random choices in Adaptive LMH are shown for 1000, 10 000, and 100 000 samples.

As can be seen in the plots, Adaptive LMH (black) exhibits faster convergence over the whole range of evaluation, requiring half as many samples as LMH (cyan) to achieve the same approximation, with the median of LMH above the 75% quantile of Adaptive LMH.

In addition, the bar plots show unit rewards and sample counts for different random choices, providing an insight on the adaptive behavior of AdLMH.

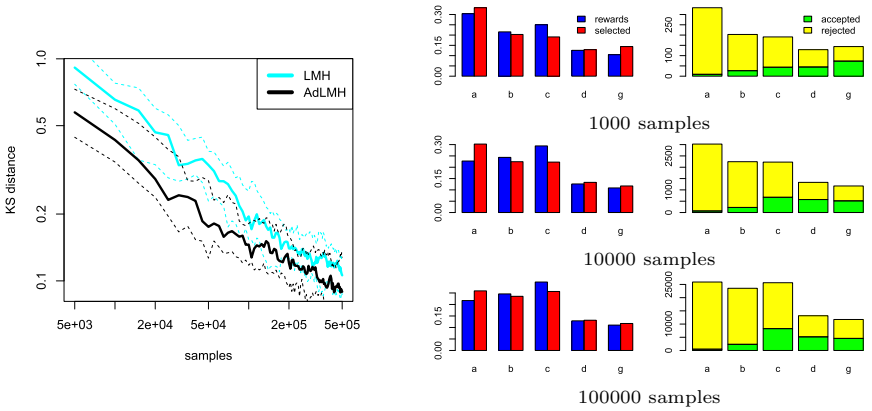

Fig. 2. Gaussian process hyperparameter estimation

On the left-hand bar plots, red bars are normalized unit rewards, and blue bars are normalized sample counts. On the right-hand bar plots, the total height of a bar is the total sample count, with green section corresponding to the accepted, and yellow to the rejected samples. At 1 000 samples, the unit rewards have not yet converged, and exploration supersedes exploitation: random choices with lower acceptance rate are selected more often (choices 7, 8 and 13 corresponding to states 6, 7 and 12). At 10 000 samples, the unit rewards become close to their final values, and choices 1 and 18, immediately affecting the predicted states, are selected more often. At 100 000 samples, the unit rewards converge, and the sample counts correspond closely to the equilibrium state outlined in Lemma 1.

The second case study is estimation of hyperparameters of a Gaussian Process. We define a Gaussian Process of the form

$$f \sim \mathcal{GP}(m, k),$$
$$\text{where } m(x) = ax^2 + bx + c, \quad k(x, x') = de^{-\frac{(x-x')^2}{2g}}.$$

The process has five hyperparameters, a, b, c, d, g. The program infers the posterior values of the hyperparameters by maximizing marginal likelihood of 6 observations $(0.0, 0.5)$, $(1.0, 0.4)$, $(2.0, 0.2)$, $(3.0, -0.05)$, $(4.0, -0.2)$, and $(5.0, 0.1)$. Parameters a, b, c of the mean function are predicted. Maximum of KS distances between inferred distributions of each of the predicted parameters and an approximation of the target distributions is shown in Figure 2. The approximation was obtained by running LMH with 2 000 000 samples per restart and 50 restarts, and then taking each 100th sample from the last 10 000 samples of each restart, 5000 samples total. Just as for the previous case study, bar plots of unit rewards and sample counts are shown for 1000, 10 000, and 100 000 samples.

Here as well, Adaptive LMH (black) converges faster over the whole range of evaluation, outperforming LMH by a factor 2 over the first 50 000 samples. Bar plots of unit rewards and sample counts for different number of choices,

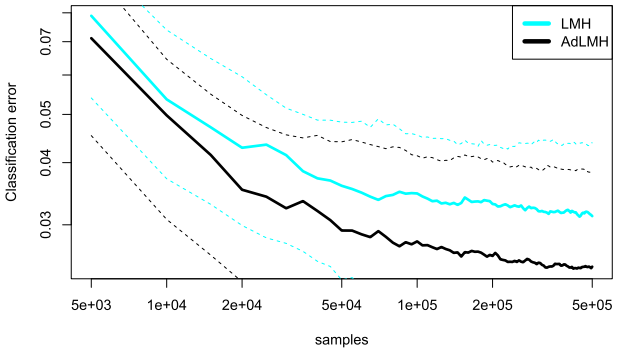

Fig. 3. Logistic regression on Iris dataset.

again, show the dynamics of sample allocation among random choices. Choices a, b, and c are predicted, while choices d and g are required for inference but not predicted. Choice a has the lowest acceptance rate (ratio between the total height of the bar and the green part on the right-hand bar plot), but the unit reward is close the unit reward of choices b and c. At 1 000 samples, choice a is selected with the highest probability. However, close to the converged state, at 100 000 samples, choices a, b, and c are selected with similar probabilities. At the same time, choices 4 and 5 are selected with a lower probability. Both the exploration-exploitation dynamics for choices a–c and probability matching of selection probabilities among all choices secure improved convergence.

The third case study involves a larger amount of data observed during each simulation of a probabilistic program. We use the well-known Iris dataset [9] to fit a model of classifying a given flower as of the species Iris setosa, as opposite to either Iris virginica or Iris versicolor. Each record in the dataset corresponds to an observation. For each observation, we define a feature vector $\boldsymbol{x}$ and an indicator variable z_i, which is 1 if and only if the observation is of an Iris setosa. We fit the model with five regression coefficients $\beta_1, \ldots, \beta_5$, defined as

$$\begin{aligned} \sigma^2 &\sim \text{InvGamma}(1, 1), \\ \beta_j &\sim \text{Normal}(0, \sigma), \\ p(z_i = 1) &= \frac{1}{1 + e^{-\beta^T x}}. \end{aligned}$$

To assess the convergence, we perform shuffle split leave-2-out cross validation, selecting one instance belonging to the species Iris setosa and one belonging to a different species for each run of the inference algorithm. The classification error is shown in Figure 3 over 100 runs of LMH and Adaptive LMH. The results are consistent with other case studies: Adaptive LMH exhibits a faster convergence rate, requiring half as many samples to achieve the same classification accuracy as LMH.

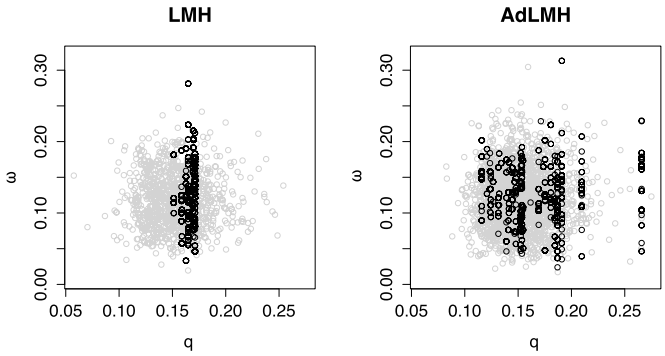

Fig. 4. Kalman filter, 500 samples after 10 000 samples of burn-in.

As a final case study we consider a linear dynamical system (i.e. a Kalman smoothing problem) that was previously described in [12]

$$\boldsymbol{x}_t \sim \mathrm{Norm}(\boldsymbol{A} \cdot \boldsymbol{x}_{t-1}, \boldsymbol{Q}), \qquad \boldsymbol{y}_t \sim \mathrm{Norm}(\boldsymbol{C} \cdot \boldsymbol{x}_t, \boldsymbol{R}).$$

In this problem we assume that 16-dimensional observations $\boldsymbol{y}_t$ are conditioned on 2-dimensional latent states $\boldsymbol{x}_t$. We impose additional structure by assuming that the transition matrix $\boldsymbol{A}$ is a simple rotation with angular velocity ω, whereas the transition covariance $\boldsymbol{Q}$ is a diagonal matrix with coefficient q,

$$\boldsymbol{A} = \begin{bmatrix} \cos\omega & -\sin\omega \\ \sin\omega & \cos\omega \end{bmatrix}, \qquad \boldsymbol{Q} = \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix}.$$

We predict posterior values for ω, and q in a setting where $\boldsymbol{C}$ and $\boldsymbol{R}$ are assumed known, under mildly informative priors $\omega \sim \mathrm{Gamma}(10, 2.5)$ and $q \sim \mathrm{Gamma}(10, 100)$. Posterior inference is performed conditioned on a simulated sequence $\boldsymbol{y}_{1:T}$ of $T = 100$ observations, with $\omega^* = 4\pi/T$, and $q^* = 0.1$. The observation matrix $\boldsymbol{C}$ and covariance $\boldsymbol{R}$ are sampled row-wise from symmetric Dirichlet distributions with parameters $c = 0.1$, and $r = 0.01$ respectively.

Figure 4 shows a qualitative evaluation of the mixing rate in the form of 500 consecutive samples (ω, q) from an LMH and AdLMH chain after 10 000 samples of burn-in. The LMH sequence exhibits good mixing over ω but is strongly correlated in q, whereas the AdLMH sequence obtains a much better coverage of the space.

To summarize, Adaptive LMH consistently attained faster convergence than LMH, measured by differences between the ongoing output distribution of the random program and the target independently obtained distribution, assessed using various metrics. Variable selection probabilities computed by Adaptive LMH are dynamically adapted during the inference, combining exploration of the model represented by the probabilistic program and exploitation of influence of random variables on program output.

6 Contribution and Future Work

In this paper we introduced a new algorithm, Adaptive LMH, for approximate inference in probabilistic programs. This algorithm adjusts sampling parameters based on the output of the probabilistic program in which the inference is performed. Contributions of the paper include

- A scheme of rewarding random choice based on program output.
- An approach to propagation of choice rewards to MH proposal scheduling parameters.
- An application of this approach to LMH, where the probabilities of selecting each variable for modification are adjusted.

Adaptive LMH was compared to LMH, its non-adaptive counterpart, and was found to consistently outperform LMH on several probabilistic programs, while still being almost as easy to implement. The time cost of additional computation due to adaptation was negligible.

Although presented in the context of a particular sampling algorithm, the adaptation approach can be extended to other sampling methods. We believe that various sampling algorithms for probabilistic programming can benefit from output-sensitive adaptation. Additional potential for improvement lies in acquisition of dependencies between predicted expressions and random variables. Exploring alternative approaches for guiding exploration-exploitation compromise, in particular, based on Bayesian inference, is another promising research direction.

Overall, output-sensitive approximate inference appears to bring clear advantages and should be further explored in the context of probabilistic programming models and algorithms.

Acknowledgments. This work is supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement number FA8750-14-2-0004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation heron. The views and conclusions contained herein are those of the authors and should be not interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U.S. Air Force Research Laboratory or the U.S. Government.

References

1. Andrieu, C., Thoms, J.: A tutorial on adaptive MCMC. Statistics and Computing **18**(4), 343–373 (2008)
2. Atchadé, Y., Fort, G., Moulines, E., Priouret, P.: Adaptive markov chain monte carlo: theory and methods. In: Barber, D., Cemgil, A.T., Chiappa, S. (eds.) Bayesian Time Series Models, pp. 32–51. Cambridge University Press (2011)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the Multiarmed Bandit problem. Machine Learning **47**(2–3), 235–256 (2002)

4. Bai, Y., Roberts, G.O., Rosenthal, J.S.: On the containment condition for adaptive Markov chain Monte Carlo algorithms. Advances and Applications in Statistics **21**(1), 1–54 (2011)
5. Gamerman, D., Lopes, H.F.: Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference. Chapman and Hall/CRC (2006)
6. Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: UAI, pp. 220–229 (2008)
7. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: ICSE (FOSE track) (2014)
8. Łatuszyński, K., Roberts, G.O., Rosenthal, J.S.: Adaptive Gibbs samplers and related MCMC methods. Annals of Applied Probability **23**(1), 66–98 (2013)
9. Lauritzen, S.: Graphical Models. Clarendon Press (1996)
10. Levine, R.A., Yu, Z., Hanley, W.G., Nitao, J.J.: Implementing componentwise hastings algorithms. Computational Stastistics & Data Analysis **48**(2), 363–389 (2005)
11. Mansinghka, V.K., Selsam, D., Perov, Y.N.: Venture: a higher-order probabilistic programming platform with programmable inference. CoRR abs/1404.0099 (2014)
12. van de Meent, J.W., Yang, H., Mansinghka, V., Wood, F.: Particle Gibbs with Ancestor Sampling for Probabilistic Programs. In: AISTATS, pp. 986–994 (2015)
13. Nori, A.V., Hur, C.K., Rajamani, S.K., Samuel, S.: R2: An efficient mcmc sampler for probabilistic programs. In: AAAI, pp. 2476–2482 (2014)
14. Roberts, G.O., Rosenthal, J.S.: Coupling and ergodicity of adaptive MCMC. Journal of Applied Probability **44**, 458–475 (2007)
15. Roberts, G.O., Rosenthal, J.S.: Examples of adaptive MCMC. Journal of Computational and Graphical Statistics **18**(2), 349–367 (2009)
16. Wingate, D., Stuhlmüller, A., Goodman, N.D.: Lightweight implementations of probabilistic programming languages via transformational compilation. In: AISTATS, pp. 770–778 (2011)
17. Wood, F., van de Meent, J.W., Mansinghka, V.: A new approach to probabilistic programming inference. In: AISTATS, pp. 1024–1032 (2014)
18. Yang, L., Hanrahan, P., Goodman, N.D.: Generating efficient MCMC kernels from probabilistic programs. In: AISTATS, pp. 1068–1076 (2014)