Automatic adaptation of MCMC algorithms

Dao Nguyen[1], Perry de Valpine[2], Yves Atchade[3],
Daniel Turek[4], Nicholas Michaud[2,5] and Christopher Paciorek[5]
[1] Departments of Mathematics, University of Mississippi, Oxford
[2] Department of Environmental Science, Policy, and Management
University of California, Berkeley
[3] Department of Statistics, University of Michigan, Ann Arbor
[4] Department of Mathematics & Statistics, Williams College, Williamstown
[5] Department of Statistics, University of California, Berkeley

Abstract

Markov chain Monte Carlo (MCMC) methods are ubiquitous tools for simulation based inference in many fields but designing and identifying good MCMC samplers is still an open question. This paper introduces a novel MCMC algorithm, namely, Auto Adapt MCMC. For sampling variables or blocks of variables, we use two levels of adaptation where the inner adaptation optimizes the MCMC performance within each sampler, while the outer adaptation explores the valid space of kernels to find the optimal samplers. We provide a theoretical foundation for our approach. To show the generality and usefulness of the approach, we describe a framework using only standard MCMC samplers as candidate samplers and some adaptation schemes for both inner and outer iterations. In several benchmark problems, we show that our proposed approach substantially outperforms other approaches, including an automatic blocking algorithm, in terms of MCMC efficiency and computational time.

Keywords: adaptive MCMC, MCMC efficiency, integrated autocorrelation time, mixing.

1 Introduction

Markov chain Monte Carlo (MCMC) has become a widely used approach for simulation from an arbitrary distribution of interest, typically a Bayesian posterior distribution, known as the target distribution. MCMC really represents a family of sampling methods. Generally speaking, any new sampler that can be shown to preserve the ergodicity of the Markov chain such that it converges to the target distribution is a member of the family and can be combined with other samplers as part of a valid MCMC kernel. The key to MCMC's success is its simplicity and applicability. In practice, however, it sometimes needs a lot of non-trivial tuning to work well (Haario et al., 2005).

To deal with this problem, many adaptive MCMC algorithms have been proposed (Gilks et al., 1998; Haario et al., 2001; Andrieu and Robert, 2001; Sahu et al., 2003). These allow parameters of the MCMC kernel to be automatically tuned based on previous samples. This breaks the Markovian property of the chain so has required special schemes and proofs that the resulting chain will converge to the target distribution (Andrieu and Moulines, 2003; Atchadé et al., 2005; Andrieu and Atchadé, 2006). Under some weaker and easily verifiable conditions, namely "diminishing adaptation" and "containment", Rosenthal and Roberts (2007) proved ergodicity of adaptive MCMC and proposed many useful samplers.

It is important to realize, however, that such adaptive MCMC samplers address only a small aspect of a much larger problem. A typical adaptive MCMC sampler will approximately optimize performance given the kind of sampler chosen in the first place, but it will not optimize among the variety of samplers that could have been chosen. For example, an adaptive random walk Metropolis-Hastings sampler will adapt the scale of its proposal distribution, but that adaptation won't reveal whether an altogether different kind of sampler would be more efficient. In many cases it would, and the exploration of different sampling strategies often remains a human-driven trial-and-error affair.

Here we present a method for a higher level of MCMC adaptation. The adaptation explores a potentially large space of valid MCMC kernels composed of different samplers. One starts with an arbitrary set of candidate samplers for each dimension or block of dimensions in the target distribution. The main idea is to iteratively try different candidates that compose a valid MCMC kernel, run them for a relatively short time, generate the next set of candidates based on the results thus far, and so on. Since relative performance of different samplers is specific to each model and even to each computing environment, it is doubtful whether there is a universally optimal kind of sampler. Hence we view the choice of efficient samplers for a particular problem as well-suited to empirical determination via computation.

The goal of computationally exploring valid sampler combinations in search of an efficient model-specific MCMC kernel raises a number of challenges. First, one must prove that the samples collected as the algorithm proceeds indeed converge to the target distribution, even when some of the candidate samplers are internally adaptive, such as conventional adaptive random walk samplers. We provide such a proof for a general framework.

Second, one must determine efficient methods for exploring the very large, discrete space of valid sampler combinations. This is complicated by a combinatorial explosion, which is exacerbated by the fact that any multivariate samplers can potentially be used for arbitrary blocks of model dimensions. Here we take a practical approach to this problem, setting as our goal only to show basic schemes that can yield substantial improvements in useful time frames. Future work can aim to develop improvements within the general framework presented here. We also limit ourselves to relatively simple candidate samplers, but the framework can accommodate many more choices.

Third, one must determine how to measure the efficiency of a particular MCMC kernel for each dimesion and for the entire model, in order to have a metric to seek to optimize. As a first step, it is vital to realize that there can be a tradeoff between good mixing and computational speed. When considering adaptation within one kind of sampler, say adaptive random walk, one can roughly assume that computational cost does not depend on the proposal scale, and hence mixing measured by integrated autocorrelation time, or the related effective sample size, is a sensible measure of efficiency. But when comparing two samplers with very different computational costs, say adaptive random walk and slice samplers, good mixing may or may not be worth its computational cost. Random walk samplers may mix more slowly than slice samplers on a per iteration basis, but they do so at higher computational speed because slice samplers can require many evaluations of model density functions. Thus the greater number of random walk iterations per unit time could outperform the slice sampler. An additional issue is that different dimensions of the model may mix at different rates, and often the slowest-mixing dimensions limit the validity of all results (Turek et al., 2016). In view of these considerations, we define MCMC efficiency as the effective sample size per computation time and use that as a metric of performance per dimension. Performance of an MCMC kernel across all dimensions is defined as the minimum efficiency among all dimensions.

The rest of the paper is organized as follows. Section 2 begins with a general theoretical framework for Auto Adapt MCMC, then extends these methods to a specific Auto Adapt algorithm involving block MCMC updating. Section 3 presents an example algorithm that fits within the framework, and provides some explanations on its details. Section 4 then outlines some numerical examples comparing the example algorithm with existing algorithms for a variety of benchmark models. Finally, section 5 concludes and discusses some future research directions.

2 A general Auto Adapt MCMC

In this section, we present a general Auto Adapt MCMC algorithm and give theoretical results establishing its correctness.

Let $\mathcal{X}$ be a state space and π the probability distribution on $\mathcal{X}$ that we wish to sample from. Let $\mathcal{I}$ be a countable set (this set indexes the discrete set of MCMC kernels we wish to choose from). For $\iota \in \mathcal{I}$, let Θ_ι be a parameter space (for all practical purposes, we can assume that this is some subset of some Euclidean space $\mathbb{R}^m$). For $\iota \in \mathcal{I}$ and $\theta \in \Theta_\iota$, let $P_{\iota,\theta}$ denote a Markov kernel on $\mathcal{X}$ with invariant distribution π. We set $\bar{\Theta} = \bigcup_{\iota \in \mathcal{I}} \{\iota\} \times \Theta_\iota$ the adaptive MCMC parameter space. We want to build a stochastic process (an adaptive Markov chain) $\{(X_n,\ \iota_n,\ \theta_n),\ n\ \geq 0\}$ on $\mathcal{X} \times \bar{\Theta}$ such that as $n \to \infty$, the distribution of X_n converges to π, and a law of large numbers holds. We call ι the external adaptation parameter and θ the internal adaptation parameter.

We will follow the general adaptive MCMC recipe of Roberts and Rosenthal (2009). Assume that any internal adaptation on Θ_ι is done using a function $H_\iota : \Theta_\iota \times \mathcal{X} \to \Theta_\iota$, and an "internal clock" sequence $\{\gamma_n,\ n\ \geq 0\}$ such that $\lim_{n\to\infty} \gamma_n = 0$. The function H_ι depends on γ_n and updates parameters for internal adaptation. Also, let $\{p_k,\ k \geq 1\}$ be a sequence of numbers $p_k \in (0,1)$ such that $\lim_{k\to\infty} p_k = 0$. p_k will be the probability of performing external adaptation at external iteration k. During the algorithm we will also keep track of two variables: κ_n, the number of external adaptations performed up to step

n; and τ_n, the number of iterations between n and the last time an external adaptation is performed. These two variables are used to manage the internal clock based on external iterations, which in most situations can simply be the number of adaptation steps. We build the stochastic process $\{(X_n,\ \iota_n,\ \theta_n),\ n\ \geq 0\}$ on $\mathcal{X} \times \bar{\Theta}$ as follows.

1. We start with $\kappa_0 = \tau_0 = 0$. We start also with some $X_0 \in \mathcal{X},\ \iota_0 \in \mathcal{I}$, and $\theta_0 \in \Theta_{\iota_0}$.
2. At the n-th iteration, given $\mathcal{F}_n \overset{def}{=} \sigma\{(X_k,\ \iota_k,\ \theta_k),\ k \leq n\}$, and given $\kappa_n,\ \tau_n$:
   (a) Draw $X_{n+1} \sim P_{\iota_n,\theta_n}(X_n,\ \cdot)$.
   (b) Independently of $\mathcal{F}_n$ and X_{n+1}, draw $B_{n+1} \sim \text{Bern}(p_{n+1}) \in \{0,1\}$.
      i. If $B_{n+1} = 0$, there is no external adaptation: $\iota_{n+1} = \iota_n$. We update κ_n and τ_n:
$$\kappa_{n+1} = \kappa_n,\ \tau_{n+1} = \tau_n + 1. \tag{1}$$
      Then we perform an internal adaptation: set $c_{n+1} = \kappa_{n+1} + \tau_{n+1}$, and compute
$$\theta_{n+1} = \theta_n + \gamma_{c_{n+1}} H_{\iota_n}(\theta_n,\ X_{n+1}). \tag{2}$$
      Note that the internal adaptation interval could vary between iterations.
      ii. If $B_{n+1} = 1$, then we do an external adaptation: we choose a new ι_{n+1}. And we choose a new value $\theta_{n+1} \in \Theta_{\iota_{n+1}}$ based on $\mathcal{F}_n$ and X_{n+1}. Then we update κ_n and τ_n.
$$\kappa_{n+1} = \kappa_n + 1,\ \tau_{n+1} = 0. \tag{3}$$

For this Auto Adapt MCMC algorithm to be valid we must show that it satisfies three assumptions:

1. For each $(\iota,\ \theta) \in \bar{\Theta},\ P_{\iota,\theta}$ has invariant distribution π.
2. (diminishing adaptation):
$$\triangle_{n+1} \overset{def}{=} \sup_{x\in\mathcal{X}} \|P_{\iota_n,\theta_n}(x,\ \cdot) - P_{\iota_{n+1},\theta_{n+1}}(x,\ \cdot)\|_{\text{TV}}$$
   converges in probability to zero, as $n \to \infty$,

3. (containment): For all $\epsilon > 0$, the sequence $\{M_\epsilon(\iota_n,\ \theta_n,\ X_n)\}$ is bounded in probability, where
$$M_\epsilon(\iota,\ \theta,\ x) \overset{def}{=} \inf\{n\ \geq 1 : \|P^n_{\iota,\theta}(x,\ \cdot) - \pi\|_{\mathrm{TV}} \leq \epsilon\}.$$

Remark 1. *Here the first assumption holds by construction. We will show that by the design, our Auto Adapt algorithm satisfies the diminishing adaptation.*

For $\iota \in \mathcal{I},\ \theta,\ \theta' \in \Theta_\iota$, define

$$D_\iota(\theta,\ \theta')\ \overset{def}{=}\ \sup_{x\in\mathcal{X}} \|P_{\iota,\theta}(x,\ \cdot) - P_{\iota,\theta'}(x,\ \cdot)\|_{\mathrm{TV}}.$$

Proposition 1. *Suppose that $\mathcal{I}$ is finite, and for any $\iota \in \mathcal{I}$, the adaptation function H_ι is bounded, and there exists $C < \infty$ such that*

$$D_\iota(\theta,\ \theta') \leq C\|\theta - \theta'\|.$$

Then the diminishing adaptation holds.

Proof. We have

$$\begin{aligned}
\mathrm{E}(\triangle_{n+1}) &= p_{n+1}\mathrm{E}(\triangle_{n+1}|B_{n+1} = 1) + (1 - p_{n+1})\mathrm{E}(\triangle_{n+1}|B_{n+1} = 0),\\
&\leq 2p_{n+1} + \mathrm{E}(\triangle_{n+1}|B_{n+1} = 0),\\
&= 2p_{n+1} + \mathrm{E}\ [D_{\iota_n}(\theta_n,\ \theta_{n+1})],\\
&\leq 2p_{n+1} + C\mathrm{E}\ [\|\theta_{n+1} - \theta_n\|],\\
&\leq 2p_{n+1} + C_1\gamma_{c_{n+1}},
\end{aligned}$$

where $c_{n+1} = \kappa_{n+1} + \tau_{n+1}$. It is easy to see that $c_n \to \infty$ as $n \to \infty$. The result follows since $\lim_{n\to\infty} p_n = \lim_{n\to\infty} \gamma_n = 0$. □

In general, the containment condition is more challenging than the diminishing adaptation condition. This technical assumption might be harder to satisfy and might not even be necessary sometimes (Rosenthal and Roberts, 2007). However, we still use simultaneous uniform ergodicity, a sufficient containment condition to simplify theory and to concentrate more on designing efficient algorithm.

Definition 1. *The family $\{P_{\iota,\theta} : (\iota, \theta) \in \bar{\Theta}\}$ has simultaneous uniform ergodicity (SUE) if for all $\epsilon > 0$, there is $N = N(\epsilon) \in \mathbb{N}$ so that $\|P^N_{\iota,\theta}(x, \ \cdot) - \pi(\cdot)\| \leq \epsilon$ for all $x \in \mathcal{X}$ and $(\iota, \theta) \in \bar{\Theta}$.*

Proposition 2. *(Theorem 1 of Rosenthal and Roberts (2007)). SUE implies containment.*

Remark 2. *It is possible to use weaker condition such as simultaneous geometric ergodicity or simultaneous polynormal ergodicity instead of simultaneous uniform ergodicity to imply containment condition but for the purpose of introducing the Auto Adapt algorithm, we do not pursue them here.*

3 Example algorithms

We present one specific approach as an example of an Auto Adapt algorithm. Our approach to "outer adaptation" will be to identify the "worst-mixing dimension" (i.e., some parameter or latent state of the statistical model) and update the kernel by assigning different sampler(s) for that dimension. To explain the method, we will give some terminology for describing our algorithm. In particular, we will define a valid kernel, MCMC efficiency, and worst-mixing dimension. We will define a set of candidate samplers for a given dimension, which could include scalar samplers or block samplers. In either case, a sampler may also have internal adaptation for each parameter or combination. To implement the internal clock of each sampler (c_n of the general algorithm), we need to formulate all internal adaptation in the framework using equation 2. We use P (without subscripts) in this section to represent $P_{\iota,\theta}$ of the general theory, so the kernel and parameters are implicit.

3.1 Valid kernel

Assume our model of interest is $\mathcal{M}$, which could be represented as a graphical model where vertices or nodes represent states or data while edges represent dependencies among them. Here we are using "state" as Bayesians do to mean any dimension of the model to be sampled by MCMC, including model parameters and latent states. We denote the set of all dimensions of the target distribution, as $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$. Since we will construct a new MCMC kernel as an ordered set of samplers at each outer iteration, it is useful to define requirements for a kernel to be valid. We require that each kernel, if used on its own, would be a valid MCMC to sample from the target distribution $\pi(X)$, $X \in \mathcal{X}$ (typically defined from Bayes' Rule as the conditional distribution of states given the data). This is the case if it satisfies the detailed balance equation, $\pi = P\pi$.

In more detail, we need to ensure that a new MCMC kernel does not omit some subspace of $\mathcal{X}$ from mixing. Denote the kernel P as a sequence of (new choice of c_{n+1}) samplers P_i, $i = 1 \ldots j$, such that $P = P_j P_{j-1} \ldots P_1$. By some abuse of language, P is a valid kernel if each sampler P_i operates on a non-empty subset b_i of $\mathcal{X}$, satisfying $\bigcup_{i=1}^{j} b_i = \mathcal{X}$.

At iteration n, assume the kernel is $P^{(n)}$ and the samples are $X_n = (X_{n,1}, \ \ldots, \ X_{n,m})$ where the set of initial values is X_0. For each dimension $\mathcal{X}_k$, $k = 1, \ldots, m$ let $\mathbf{X}_k = \{X_{0,k}, \ X_{1,k}, \ \ldots\}$ be the scalar chain of samples of $\mathcal{X}_k$.

3.2 Worst mixing state and MCMC efficiency

We define MCMC efficiency for state $\mathcal{X}_k$ from a sample of size N from kernel P as effective sample size per computation time

$$\omega_k(N, P) = \frac{N/\tau_k(P)}{t(N, P)},$$

where $t(N, P)$ is the computation time for kernel P to run N iterations (often $t(N, P) \approx Nt(1, P)$) and $\tau_k(P)$ is the integrated autocorrelation time for chain $\mathbf{X}_k$ defined as

$$\tau_k = 1 + 2 \sum_{i=1}^{\infty} \operatorname{cor}(X_{0,k}, X_{i,k}),$$

Straatsma et al. (1986). The ratio N/τ_k is the effective sample size (ESS) for state $\mathcal{X}_k$ (Roberts et al., 2001). Note that $t(N, P)$ is computation time for the entire kernel, not just samplers that update $\mathcal{X}_k$. τ_k can be interpreted as the number of effective samples per actual sample. The worst-mixing state is defined as the state with minimum MCMC efficiency among all states. Let k_{min} be the index of the worst-mixing state, that is

$$k_{min} = \arg\min_{k} \tau_k^{-1}$$

Since the worst mixing dimension will limit the validity of the entire posterior sample (Thompson, 2010), we define the efficiency of a MCMC algorithm as $\omega_{k_{min}}(N, P)$, the efficiency of the worst-mixing state of model $\mathcal{M}$.

There are several ways to estimate ESS, but we use `effectiveSize` function in the R coda package (Plummer et al., 2006; Turek et al., 2016) since this function provides a stable estimation of ESS. This method, which is based on the spectral density at frequency zero, has been proven to have the highest convergence rate, thus giving a more accurate and stable result (Thompson, 2010).

3.3 Candidate Samplers

A set of candidate samplers $\{P_j,\ j \in \mathcal{S}\}$ is a list of all possible samplers that could be used for a parameter of the model $\mathcal{M}$. These may differ depending on the parameter's characteristics and role in the model (e.g., whether there is a valid Gibbs sampler, or whether it is restricted to $[0, \infty)$). In addition to univariate candidate samplers, nodes can also be sampled by block samplers. Denote $|b|$ the number of elements of b. If $|b_i| > 1$, $P_i^{(n)}$, the sampler applied to block b_i at iteration n, is called a block sampler; otherwise it is a univariate or scalar sampler.

In the examples below we considered up to four univariate candidate samplers and three kinds of block samplers. The univariate samplers included adaptive random walk (ARW), adaptive random walk on a log scale (ARWLS) for states taking only positive real values, Gibbs samplers for states with a conjugate prior-posterior pairing, and slice samplers. The block samplers included adaptive random walk with multivariate normal proposals, automated factor slice sampler (Tibbits et al., 2014) (slice samplers in a set of orthogonal rotated coordinates), and automated factor random walk (univariate random walks in a set of orthogonal rotated coordinates). These choices are by no means exhaustive but serve to illustrate the algorithms here.

3.3.1 Block samplers and how to block

(Turek et al., 2016) suggested different ways to block the states efficiently: (a) based on correlation clustering, (b) based on model structure. Here we use the first method.

At each iteration, we use the generated samples to create the empirical posterior correlation matrix. To stabilize the estimation, all of the samples are used to compute a correlation matrix $\rho_{d\times d}$. This in turn is used to make a distance matrix $D_{d\times d}$ where $D_{i,j} = 1 - |\rho_{i,j}|$ for $i \neq j$ and $D_{i,i} = 0$ for every i, j in $1, \ldots, d$. To ensure minimum absolute pairwise correlation between clusters, we construct a hierarchical cluster tree from the distance matrix D (Everitt et al. (2011) chapter 4). Given a selected height, we cluster the hierarchical tree into distinct groups of states. Different parts of the tree may have different optimal heights for forming blocks. Instead of using a global height to cut the tree, we only choose a block that contains the worst-mixing state from the cut and keep the other nodes intact. Adaptively, at each outer iteration, the algorithm will try to obtain a less correlated cluster for a chosen block sampler to improve on the efficiency. In our implementation, we use the R function `hclust` to build the hierarchical clustering tree with "complete linkage" from the distance matrix D. By construction, the absolute correlation between states within each group is at least $1 - h$ for h in $[0, 1]$. We then use the R function `cutree` to choose a block that contains the worst-mixing state. This process is justified in the sense that the partitioning adapts according to the model structure through the posterior correlation. The details and validity of the block sampling in our general framework are provided in Appendix A.

3.4 How to choose new samplers

To choose new samplers to compose a new kernel, we determine the worst-mixing state and choose randomly from candidate samplers to replace whatever sampler was updating it in the previous kernel while keeping other samplers the same. There are some choices to make when considering a block sampler. If the worst-mixing parameter is x, and the new kernel will use a block sampler for x together with one or more parameters y, we can either keep the current sampler(s) used for y or remove them from the kernel. Future work can consider other schemes such as changing group of samplers together based on model structure.

3.5 Internal clock variables

In the algorithm 1, θ represents the internal adaptation parameter of a particular sampler and c represents its internal clock. In general, an internal clock variable is defined as a variable used in a sampler to determine the size of internal adaptation steps such that any internal adaptation would converge in a typical MCMC setting. An example of an internal clock variable is a number of internal iterations that have occurred. To use a sampler in the general framework, we need to establish what are its internal adaptation and clock variables. A few examples of internal adaptation variables of different samplers are summarized as follows:

- For adaptive random walk: proposal scale is used.
- For block adaptive random walk: proposal scale and covariance matrix are used.
- For automated factor slice sampler: covariance matrix (or equivalent, i.e. coordinate rotation) is used.
- For automated factor random walk: covariance matrix (ditto) and proposal scales for each rotated coordinate axis are used.

These internal adaption variables are set to default initial values when its sampler is first used. After that, they are retained along with internal clock variables so that whenever we revisit a sampler, we will used the stored values to set up this sampler. This setting guarantees the diminishing adaption property, which is essential for the convergence of the algorithm. Pseudo-code for Auto Adapt MCMC is given in Algorithm 1.

4 Examples

In this section, we evaluate our algorithm on some benchmark examples and compare them to different MCMC algorithms. In particular, we compare our approach to the following MCMC algorithms.

- All Scalar algorithm: Every dimension is sampled using an adaptive scalar normal random walk sampler.
- All Blocked algorithm: All dimensions are sampled in one adaptive multivariate normal random walk sampler.

Algorithm 1 Auto Adapt MCMC

```
Input:
    Bayesian model with initial state (including latent variables) $X_0$
    $\{p_n, n \in \mathbb{N} | p_n \in (0,1), \lim_n p_n = 0\}$, maximum iteration $M$
    Candidate samplers $\{P_j, j \in \mathcal{S}\}$
    $P_{\iota_0,\theta_0}$ := ordered set of initial samplers $\{P_j^{(0)}\}_{j\in\mathcal{S}}$ from Bayesian model
Output:
    An ordered set of samplers $\{P_{i^*}\}_{i^*\in\mathcal{S}}$ with the best MCMC efficiency so far
 1: Initialize EFF, $\mathrm{EFF}_{\mathrm{best}}$, $n$, $\kappa_0$, $\tau_0$, $c_0$ to 0    ▷ Denote MCMC efficiency EFF
 2: while (EFF $\geq \mathrm{EFF}_{\mathrm{best}}$) or ($n < M$) do
 3:     Sample $N$ samples from the current sampler set $\{P_j^{(n)}\}_{j\in\mathcal{S}}$
 4:     Store internal clocks $c_n$ and adaption variables $\theta_n$ for each sampler    ▷ Section 3.5
 5:     Compute $\mathrm{EFF_k} = \omega_k(N,P) = \frac{N/\tau_k(P)}{t(N,P)}$    ▷ $k$ is an index of parameters
 6:     Identify $k_{min} = \arg\min_k \tau_k^{-1}$, $\mathrm{EFF} = \mathrm{EFF_{k_{min}}}$    ▷ See Section 3.2
 7:     if (EFF $\geq \mathrm{EFF}_{\mathrm{best}}$) then
 8:         Set $\{P_{i^*}\}_{i^*\in\mathcal{S}} = \{P_i^{(n)}\}_{i\in\mathcal{S}}$
 9:         Set $\mathrm{EFF}_{\mathrm{best}} = \mathrm{EFF}$
10:     else
11:         Set $\{P_i^{(n)}\}_{i\in\mathcal{S}} = \{P_{i^*}\}_{i^*\in\mathcal{S}}$
12:     Draw $B_{n+1} \sim \mathrm{Bern}(p_{n+1}) \in \{0,1\}$
13:     if $B_{n+1} = 0$ then
14:         $\kappa_{n+1} = \kappa_n$, $\tau_{n+1} = \tau_n + 1$, $c_{n+1} = \kappa_{n+1} + \tau_{n+1}$
15:     else
16:         $\kappa_{n+1} = \kappa_n + 1$, $\tau_{n+1} = 0$, $c_{n+1} = \kappa_{n+1} + \tau_{n+1}$
17:         Set $P_i^{(n+1)} = P_i^{(n)}, i \neq k_{\mathrm{min}}$, choose $P_{k_{\mathrm{min}}}^{(n+1)}$ from candidate samplers    ▷ See Section 3.4
18:         if ($P_{k_{\mathrm{min}}}^{(n+1)}$ has been used before) then
19:             Use $c_n$, $\theta_n$ to set up the sampler $P_{k_{\mathrm{min}}}^{(n+1)}$
20:         else
21:             Use default internal adaptation value of $P_{k_{\mathrm{min}}}^{(n+1)}$    ▷ Section 3.5
22:     Set $n = n + 1$
```

- Default algorithm: Groups of parameters arising from multivariate distributions are sampled using adaptive multivariate normal random walk samplers while parameters arising from univariate distributions are sampled using adaptive scalar normal random walk samplers. In addition, whenever the structure of model $\mathcal{M}$ permits, we assigns conjugate samplers instead of scalar normal random walk samplers.

- Auto Block algorithm: The Auto Block method (Turek et al., 2016) searches blocking schemes based on hierarchical clustering from posterior correlations to determine a highly efficient (but not necessarily optimal) set of blocks that are sampled with multivariate normal random-walk samplers. Thus, Auto Block uses only either scalar or multivariate adaptive random walk, concentrating more on partitioning the correlation matrix than trying different sampling methods. Note that the initial sampler of both the Auto Block algorithm and our proposed algorithm is the All Scalar algorithm.

All experiments were carried out using the NIMBLE package (de Valpine et al., 2017) for R (R Core Team, 2013) on a cluster using 32 cores of Intel Xeon E5-2680 2.7 Ghz with 256 GB memory. Models are coded using NIMBLE's version of the BUGS model declaration language (Lunn et al., 2000, 2012). All MCMC algorithm are written in NIMBLE, which provides user friendly interfaces in R and efficient execution in custom-generated C++, including matrix operations in the C++ Eigen library (Guennebaud et al., 2010).

To measure the performance of a MCMC algorithm, we use MCMC efficiency. MCMC efficiency depends on ESS, estimates of which can have high variance for a short Markov chain. This presents a tuning-parameter tradeoff for the Auto Adapt method: Is it better to move cautiously (in sampler space) by running long chains for each outer adaptation in order to gain an accurate measure of efficiency, or is it better to move adventurously by running short chains, knowing that some algorithm decisions about samplers will be based on noisy efficiency comparisons? In the latter case, the final samplers may be less optimal, but that may be compensated by the saved computation time. To explore this tradeoff, we try our Auto Adapt algorithm with different sample sizes in each outer adaptation and label results accordingly. For example, Auto Adapt 10K will refer to the Auto Adapt method with samples of 10,000 per outer iteration.

We present algorithm comparisons in terms of time spent in an adaptation phase, final

MCMC efficiency achieved, and the time required to obtain a fixed effective sample size (e.g., 10,000). Only Auto Block and Auto Adapt have adaptation phases. An important difference is that Auto Block did not come with a proof of valid adaptive MCMC convergence (it could be modified to work in the current framework, but we compare to the published version). Therefore, samples from its adaptation phase are not normally included in the final samples, while the adapation samples of Auto Adapt can be included.

To measure final MCMC efficiency, we conducted a single long run of length N with the final kernel of each method solely for the purpose of obtaining an accurate ESS estimate. One would not normally do such a run in an real application. The calculation of time to obtain a fixed effective sample size incorporates both adaptation time and efficiency of the final samplers. For both Auto Adapt and Auto Block, we placed them on a similar playing field by assuming for this calculation that samples are not retained from the adaptation phase, making the results conservative.

For all comparisons, we used 20 independent runs of each method and present the average results from these runs. To show the variation in runs, we present boxplots of efficiency in relation to computation time from the 20 runs of Auto Adapt. The final (right-most) boxplot in each such figures shows the 20 final efficiency estimates from larger runs. Not surprisingly, these can be lower than obtained by shorter runs. These final estimates are reported in the tables.

A public Github repository containing scripts for reproducing our results may be found at https://github.com/nxdao2000/AutoAdaptMCMC. Some additional experiments are also provided there.

4.1 Toy example: A random effect model

We consider the "litters" model, which is an original example model provided with the MCMC package WinBUGS. This model is chosen because of its notoriously slow mixing, which is due to the strong correlation between parameter pairs. It is desirable to show how much improvement can be achieved compared to other approaches on this benchmark example. The purpose of using a simple example is to establish the potential utility of the Auto Adapt approach, while saving more advanced applications for future work. In this case, we show that our algorithm indeed outperforms by a significant margin the other

Table 1: Summary results of different MCMC algorithms for the litters model. Runtime is presented as seconds, and efficiency is in units of effective samples produced per second of algorithm runtime. Time to N effective samples is computed by N/efficiency for static algorithms and that plus adaptation time for Auto Block and Auto Adapt algorithms.

Algorithms	Adapt time	Efficiency	Time to 10000 effective samples
All Blocked	0.00	0.5855	17079
Default	0.00	1.8385	5439
All Scalar	0.00	1.6870	5928
Auto Block	21.97	12.1205	847
Auto Adapt 10K	1.14	9.6393	1038
Auto Adapt 20K	2.33	11.5717	866
Auto Adapt 50K	6.03	14.3932	701

approaches. This model's specification is given following Deely and Lindley (1981) and Kass and Steffey (1989) as follows.

Suppose we observe the data in i groups. In each group, the data y_{ij}, $j = \{1, \ldots, n\}$ are conditionally independent given the parameters p_{ij}, with the observation density

$$y_{ij} \sim \text{Bin}(n_{ij}, p_{ij}).$$

In addition, assume that p_{ij} for fixed i are conditionally independent given the "hyperparameters" α_i, β_i, with conjugate density

$$p_{ij} \sim \text{Beta}(\alpha_i, \beta_i).$$

Assume that α_j, β_j follow prior densities,

$$\alpha_1 \sim \text{Gamma}(1, 0.001),$$

$$\beta_1 \sim \text{Gamma}(1, 0.001),$$

$$\alpha_2 \sim \text{Uniform}(0, 100),$$

$$\beta_2 \sim \text{Uniform}(0, 50).$$

Following the setup of Rue and Held (2005); Turek et al. (2016), we can jointly sample the top-level parameters and conjugate latent states as the beta-binomial conjugacy relationships allow the use of what Turek et al. (2016) call cross-level sampling, but, for demonstration purposes, we do not include this here.

Since the litters model mixes poorly, we run a large number of iterations (i.e. $N = 300000$) to produce stable estimates of final MCMC efficiency. We start both Auto Block and Auto Adapt algorithms with All Scalar and adaptively explore the space of all given candidate samplers. We use Auto Adapt with either 10000, 20000 or 50000 iterations per outer adaptation.

Results (Table 1) show that Auto Block generates samples with MCMC efficiency about seven-fold, six-fold and twenty-fold that of the All Scalar, Default and All Blocked methods, respectively. We can also see that as the outer adaptation sample size increases, the performance of Auto Adapt improves. Final MCMC efficiencies of Auto Adapt 10K, Auto Adapt 20K and Auto Adapt 50K are 80%, 95% and 118% of MCMC efficiency of Auto Block, respectively. In addition, the adaptation time for all cases of Auto Adapt are much shorter than for Auto Block. Combining adaptation time and final efficiency into the resulting time to 10000 effective samples, we see that in this case, larger samples in each outer iteration are worth their computational cost.

Figure 1 shows the boxplots computed from 20 independent runs on litters model of All Blocked, All Scalar, Auto Block, Default and Auto Adapt 50K algorithms. The left panel of the figure confirms that MCMC efficiency of Auto Block is well dominated that of other static adaptive algorithms. The right panel of the figure shows the MCMC efficiency of Auto Adapt 50K gradually improves with time. The right-most boxplot verifies that the MCMC efficiency of selected samplers from Auto Adapt algorithm (computed from large samples) is slightly better than that of Auto Block algorithm. Last but not least, Auto Adapt algorithms are much more efficient than Auto Block in the sense that we can keep every sample while Auto Block algorithm throws away most of the samples.

4.2 Generalized linear mixed models

Many MCMC algorithms do not scale up well for high dimensional data. To test the capabilities of our algorithm in such situations, we consider a relatively large generalized linear mixed model (GLMM) (Gelman and Hill, 2006). We make use of the Minnesota

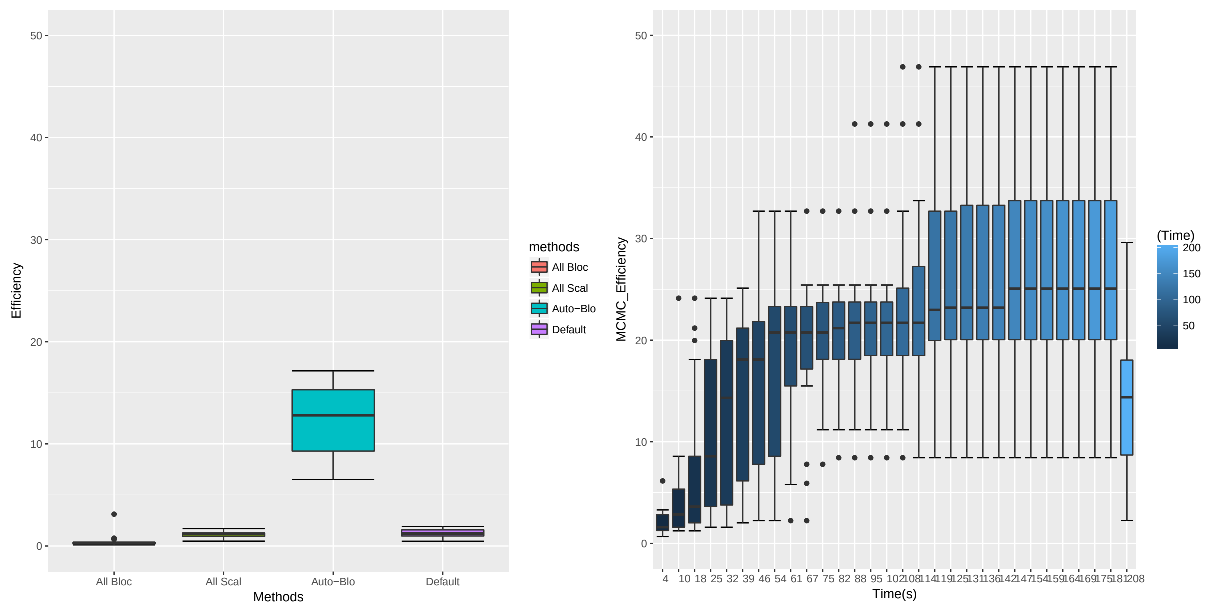

Figure 1: MCMC efficiencies of different methods for litters model. The left panel shows the box-plots of MCMC efficiencies of All Blocked, All Scalar, Auto Block and Default algorithms computed from 20 replications. The right panel show the box-plots of MCMC efficiencies of Auto Adapt 50K algorithm computed from 20 replications at each outer adaptation. The last (rightest) box-plot is computed from the chain of 300000 samples generated. The time axis show the average computational time of 20 replications.

Health Plan dataset (Waller and Zelterman, 1997) for this model following the setup of Zipunnikov and Booth (2006). Specifically, let y_{ikl} be the count for subject i ($i = 1, \ldots, 121$ senior-citizens), event k (either visited or called), period l (one of four 6-month periods). Assume that

$$y_{ikl}|u_i^{\Sigma} \sim \text{Poisson}(\mu_{ikl})$$

with log link,

$$\log \mu_{ikl} = a_0 + a_k + b_l + c_{kl} + \gamma_i + v_{ik} + \omega_{il},\ \ k = 1, 2,\ \text{and}\ l = 1, 2, 3, 4.$$

Here the fixed effect coefficients are a_k, b_l and c_{kl}. To achieve identifiability, we set $a_2 = b_4 = c_{14} = c_{21} = c_{22} = c_{23} = c_{24} = 0$. Priors for the non-zero parameters, $\beta = (a_0,\ a_1,\ b_1,\ b_2,\ b_3,\ c_{11},\ c_{12},\ c_{13})$, are:

$$a_0 \sim \text{N}(0, 0.001)$$

$$a_1 \sim \text{N}(0, 0.001)$$

Table 2: Summary results of different MCMC algorithms for GLMM model. Runtime is presented as seconds, and efficiency is in units of effective samples produced per second of algorithm runtime. Time to N effective samples is computed by N/efficiency for static algorithms and that plus adaptation time for Auto Block and Auto Adapt algorithms.

Algorithms	Adapt time	Efficiency	Time to 10000 effective samples
All Blocked	0.00	0.0031	6451613
Default	0.00	0.4641	43094
All Scalar	0.00	0.4672	42808
Auto Block	1019.35	0.8420	12896
Auto Adapt 5K	247.15	0.5289	19154
Auto Adapt 10K	465.92	0.7349	14072
Auto Adapt 20K	1017.38	0.8594	12652

$$b_l \sim \mathrm{N}(0, 0.001) \text{ for l} = 1, 2, 3$$

$$c_{1l} \sim \mathrm{N}(0, 0.001) \text{ for l} = 1, 2, 3.$$

The random effect variables are γ_i, v_{ik}, ω_{il}. Their distributions are:

$$\sigma_\gamma^2 \sim \mathrm{N}(0, 10)$$

$$\gamma_i \sim \mathrm{N}(0, \sigma_\gamma)$$

$$\sigma_v^2 \sim \mathrm{N}(0, 10)$$

$$v_{ik} \sim N(0, \sigma_\nu)$$

$$\sigma_\omega^2 \sim \mathrm{N}(0, 10)$$

$$\omega_{il} \sim \mathrm{N}(0, \sigma_\omega)$$

It should be noted that GLMM model is by far the largest example considered, containing nearly 2000 stochastic model components, which include both observations and a large number of independent random effects. Since this example is rather computationally intensive, we try our Auto Adapt algorithm with smaller numbers of iterations in each outer adaptation as well as smaller number of iteration to estimate final efficiency than we did with the litters model. Specifically, we used samples sizes of 5000, 10000 and 20000 per outer adaptation, and we used $N = 50000$ for computing final efficiency.

In this example (Table 2) All Scalar sampling produces MCMC efficiency of about 0.47, while the All Blocked algorithm, which consists of a single block sampler of dimension 858, has MCMC efficiency of approximately 0.003. In this case, All Blocked samples all 858 dimensions jointly, which requires computation time roughly three-times that of All Scalar and yields only rather low ESS. The Default algorithm performs similarly to All Scalar but they both perform much worse than Auto Block and Auto Adapt. In this example, it is clear that all Auto Adapt and Auto Block methods have dramatic improvements even when we take into account the adaptation time. Amongst these Auto methods, Auto Block performs slightly worse than Auto Adapt 20K in both computational time and MCMC efficiencies. Overall, Auto Adapt 20K appears to be the most efficient method in terms of time to 10000 effective samples. One interpretation is that Auto Adapt 20K trades off well between adaptation time and MCMC efficiency in this model.

Figure 2 shows that Auto Adapt algorithm is very competitive with the Auto Block algorithm. This comes from both the flexibility to trade off the number of outer adaptations vs. adaptive time to reach a good sampler as well as the larger space of kernels being explored. Since MCMC efficiency is highly dependent upon hierarchical model structure, using scalar and multivariate normal random walks alone, as done by the Auto Block algorithm, can be quite limiting. Auto Adapt, can overcome this limitation with the flexibility to choose different type of samplers. We will see that more strongly in the next example, where the model is more complex.

4.3 Spatial model

In this section, we consider a hierachical spatial model as the final example. We use the classical scallops dataset for this model. This dataset is chosen since we want to compare our approach with other standard approaches in the presence of spatial dependence. This data collects observations of scallop abundance at 148 locations from the New York to New Jersey coastline in 1993. It was surveyed by the Northeast Fisheries Science Center of the National Marine Fisheries Service and made publicly available at http://www.biostat.umn.edu/~brad/data/myscallops.txt. It has been analyzed many times, such as Ecker and Heltshe (1994); Ecker and Gelfand (1997); Banerjee et al.

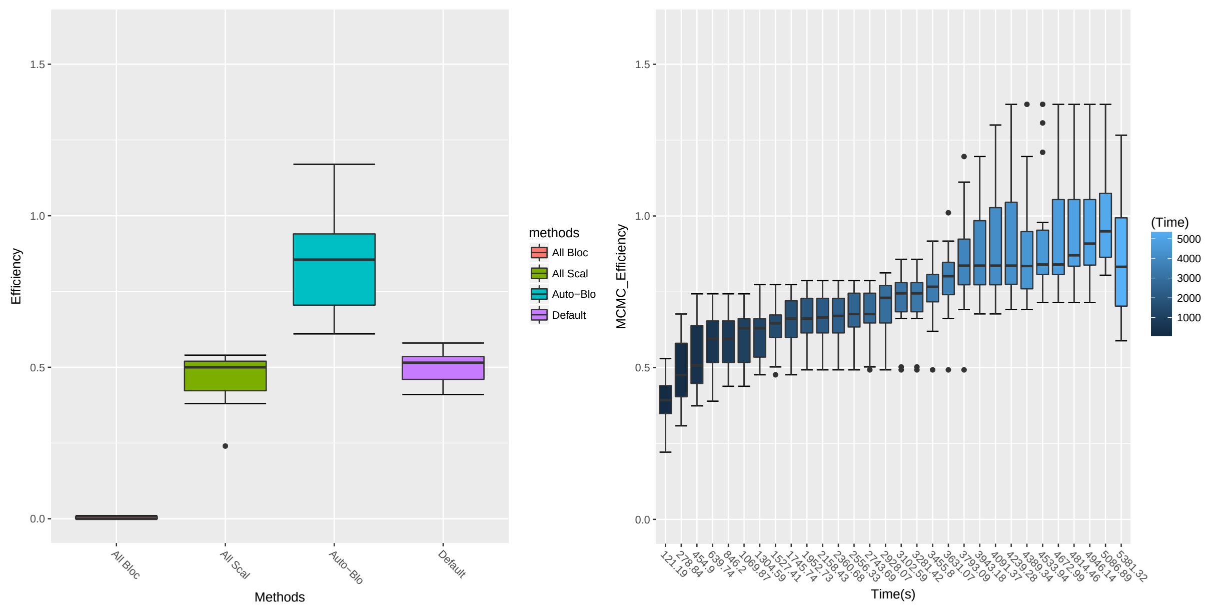

Figure 2: MCMC efficiencies of different methods for GLMMs model. The left panel shows the box-plots of MCMC efficiencies of All Blocked, All Scalar, Auto Block and Default algorithms computed from 20 replications. The right panel show the box-plots of MCMC efficiencies of Auto Adapt 20K algorithm computed from 20 replications at each outer adaptation. The last (rightest) box-plot is computed from the chain of 50000 samples generated. The time axis show the average computational time of 20 replications.

(2014) and references therein. Following Banerjee et al., assume the log-abundance $\mathbf{g} = (g_1, \ldots, g_N)$ follows a multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, defined by covariances that decay exponentially as a function of distance. Specifically, let y_i be measured scallop abundance at site i, $d_{i,j}$ be the distance between sites i and j, and ρ be a valid correlation. Then

$$\mathbf{g} \sim \mathrm{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right),$$

where each component $\Sigma_{ij} = \sigma^2 exp(-d_{i,j}/\rho)$. We model observations as $y_i \sim \text{Poisson}(\exp(g_i))$. Priors for σ and ρ are Uniform over a large range of interest. The parameters in the posterior distribution are expected to be correlated as the covariance structure induces a trade-off between σ and ρ. This can be sampled well by Auto Block algorithm, and we would like to show that our approach can achieve even higher efficiency with lower computational cost of adaptation.

This spatial model, with 858 parameters, is computationally expensive to estimate. Therefore, we will use Auto Adapt 5K, Auto Adapt 10K and Auto Adapt 20K algorithms

Table 3: Summary results of different MCMC algorithms for spatial model. Runtime is presented as seconds, and efficiency is in units of effective samples produced per second of algorithm runtime. Time to N effective samples is computed by N/efficiency for static algorithms and that plus adaptation time for Auto Block and Auto Adapt algorithms.

Algorithms	Adapt time	Efficiency	Time to 10000 effective samples
All Blocked	0.00	0.0100	1000000
Default	0.00	0.0020	5000000
All Scalar	0.00	0.1150	86956
Auto Block	19094.89	0.3565	47145
Auto Adapt 5K	2967.558	0.4420	25592
Auto Adapt 10K	6221.61	0.4565	28127
Auto Adapt 20K	11278.78	0.4948	31488

for comparison and run $N = 50000$ for estimating final efficiency.

As can be seen from the Table 3, All Blocked and Default algorithms mix very poorly, resulting in extremely low efficiencies of 0.01 and 0.002, respectively. The All Scalar algorithm, while achieving higher ESS, run slowly because large matrix calculations are needed for every univariate sampler. The Auto Block algorithm, on the other hand, selects an optimal threshold to cut the entire hierarchical clustering tree into different groups, increasing the ESS about 3 times. With a few small blocks, the computation cost of Auto Block is somewhat cheaper than All Scalar algorithm. As a result, the efficiency mean is about 3.5 times that of All Scalar. Meanwhile, our Auto Adapt 5K, 10K and 20K algorithms perform best. It should be noted that the Auto Adapt algorithm can achieve good mixing with adaptation times that are only 15.5%, 32.5% and 59% compared to the adaptation time of Auto Block. In Figure 3, while the left panel shows a distinction between Auto Block and other static algorithms, the right panel shows that Auto Adapt 20K surpasses Auto Block in just a few outer iterations, indicating substantial improvements in some models.

5 Discussion

We have proposed a general Auto Adapt MCMC algorithm. Our algorithm traverses a space of valid MCMC kernels to find an efficienct algorithm automatically. There is only

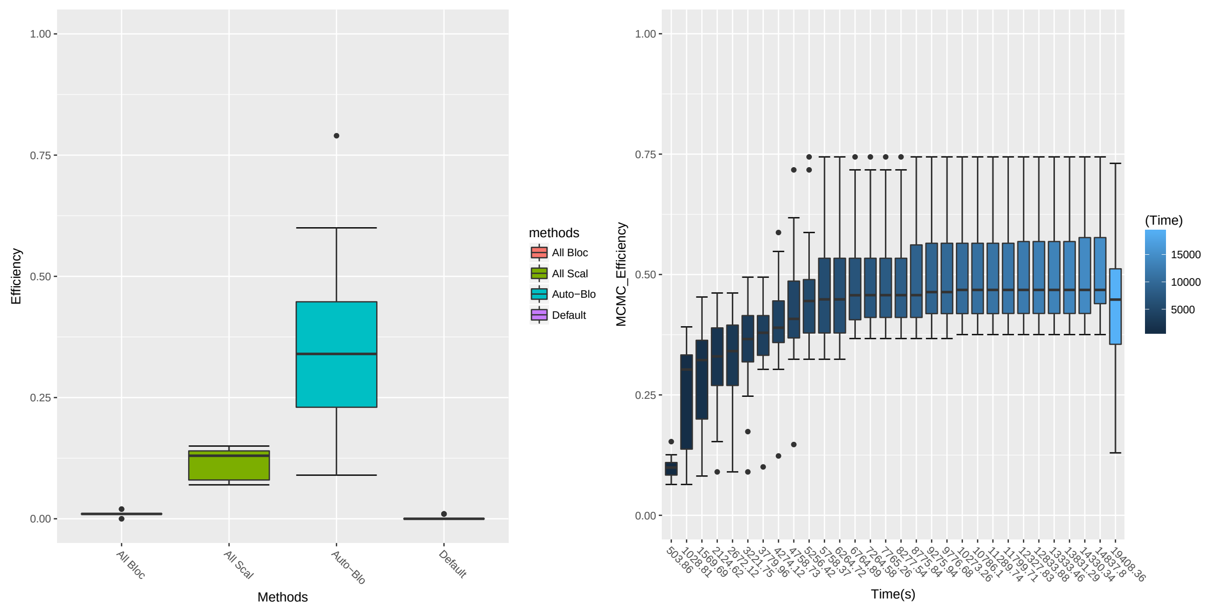

Figure 3: MCMC efficiencies of different methods for spatials model. The left panel shows the box-plots of MCMC efficiencies of All Blocked, All Scalar, Auto Block and Default algorithms computed from 20 replications. The right panel show the box-plots of MCMC efficiencies of Auto Adapt 20K algorithm computed from 20 replications at each outer adaptation. The last (rightest) box-plot is computed from the chain of 10000 samples generated. The time axis show the average computational time of 20 replications.

one previous approach, namely Auto Block sampling, of this kind that we are aware of. We have shown that our approach can substantially outperform Auto Block in some cases, and that both outperform simple static approaches. Using some benchmark models, we can observe that our approach can yield orders-of-magnitude improvements.

The comparisons presented have deliberately used fairly simple samplers as options for Auto Adapt in order to avoid comparisons among vastly different computational implementations. A major feature of our framework is that it can incorporate almost any sampler as a candidate and almost any strategy for choosing new kernels from compositions of samplers based on results so far. Samplers to be explored in the future could include auxiliary variable algorithms such as slice sampling or derivative-based sampling algorithm such as Hamiltonian Monte Carlo (Duane et al., 1987). Now that the basic framework is established and shown to be useful in simple cases, it merits extension to more advanced cases.

The Auto Adapt method can be viewed as a generalization of the Auto Block method. It is more general in the sense that it can use more kinds of samplers and explore the space of samplers more generally than the correlation-clustering of Auto Block. Thus, our framework can be considered to provide a broad class of automated kernel construction algorithms that use a wide range of sampling algorithm as components.

If block sampling is included in the space of the candidate samplers, choosing optimal blocks is important and can greatly increase the efficiency of the algorithm. For this reason, we extended the cutting of a hierarchical cluster tree to allow different cut heights on different branches (different parts of the model). This differs from Auto Block, which forms all blocks by cutting the entire tree at the same height. We also have different multivariate adaptive sampling other than random walk normal distribution such as automated factor slice sampler and automated factor random walk sampler. With these extensions, the final efficiency achieved by our algorithm specifically among blocking schemes is often substantially better and is found in a shorter time.

Beyond hierarchical clustering, there are other approaches one might consider to find efficient blocking schemes. One such approach would be to use the structure of the graph instead of posterior correlations to form blocks. This would allow conservation of calculations that are shared by some parts of the graph, whether or not they are correlated. Another future direction could be to improve how a new kernel is determined from previous results, essentially to determine an effective strategy for exploring the very high-dimensional kernel space. Finally, the tradeoff between computational cost and the accuracy of effective sample size estimates is worth further exploration.

References

Andrieu, C. and Y. F. Atchadé (2006). On the efficiency of adaptive mcmc algorithms. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, pp. 43. ACM.

Andrieu, C. and E. Moulines (2003). Ergodicity of some adaptive markov chain monte carlo algorithm. Technical report, Technical report.

Andrieu, C. and C. P. Robert (2001). Controlled mcmc for optimal sampling. *Preprint*.

Atchadé, Y. F., J. S. Rosenthal, et al. (2005). On adaptive markov chain monte carlo algorithms. *Bernoulli 11*(5), 815–828.

Banerjee, S., B. P. Carlin, and A. E. Gelfand (2014). *Hierarchical modeling and analysis for spatial data*. Crc Press.

de Valpine, P., D. Turek, C. Paciorek, C. Anderson-Bergman, D. T. Lang, and R. Bodik (2017). Programming with models: writing statistical algorithms for general model structures with NIMBLE.

Deely, J. and D. Lindley (1981). Bayes empirical bayes. *Journal of the American Statistical Association 76*(376), 833–841.

Duane, S., A. D. Kennedy, B. J. Pendleton, and D. Roweth (1987). Hybrid monte carlo. *Physics letters B 195*(2), 216–222.

Ecker, M. and J. Heltshe (1994). Geostatistical estimates of scallop abundance. *Case studies in biometry*, 107–124.

Ecker, M. D. and A. E. Gelfand (1997). Bayesian variogram modeling for an isotropic spatial process. *Journal of Agricultural, Biological, and Environmental Statistics*, 347–369.

Everitt, B. S., S. Landau, M. Leese, and D. Stahl (2011). Hierarchical clustering. *Cluster Analysis, 5th Edition*, 71–110.

Gelman, A. and J. Hill (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press.

Gilks, W. R., G. O. Roberts, and S. K. Sahu (1998). Adaptive markov chain monte carlo through regeneration. *Journal of the American statistical association 93*(443), 1045–1054.

Guennebaud, G., B. Jacob, et al. (2010). Eigen. *URl: http://eigen. tuxfamily. org*.

Haario, H., E. Saksman, and J. Tamminen (2001). An adaptive metropolis algorithm. *Bernoulli*, 223–242.

Haario, H., E. Saksman, and J. Tamminen (2005). Componentwise adaptation for high dimensional mcmc. *Computational Statistics 20*(2), 265–273.

Kass, R. E. and D. Steffey (1989). Approximate bayesian inference in conditionally independent hierarchical models (parametric empirical bayes models). *Journal of the American Statistical Association 84*(407), 717–726.

Łatuszyński, K., G. O. Roberts, J. S. Rosenthal, et al. (2013). Adaptive gibbs samplers and related MCMC methods. *The Annals of Applied Probability 23*(1), 66–98.

Lunn, D., C. Jackson, N. Best, A. Thomas, and D. Spiegelhalter (2012). *The BUGS book: A practical introduction to Bayesian analysis*. CRC press.

Lunn, D. J., A. Thomas, N. Best, and D. Spiegelhalter (2000). Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing 10*(4), 325–337.

Plummer, M., N. Best, K. Cowles, and K. Vines (2006). Coda: convergence diagnosis and output analysis for mcmc. *R news 6*(1), 7–11.

R Core Team (2013). R: A Language and Environment for Statistical Computing.

Roberts, G. O. and J. S. Rosenthal (2009). Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics 18*(2), 349–367.

Roberts, G. O., J. S. Rosenthal, et al. (1998). Two convergence properties of hybrid samplers. *The Annals of Applied Probability 8*(2), 397–407.

Roberts, G. O., J. S. Rosenthal, et al. (2001). Optimal scaling for various metropolis-hastings algorithms. *Statistical science 16*(4), 351–367.

Rosenthal, J. and G. Roberts (2007). Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of Applied Probablity 44*, 458–475.

Rue, H. and L. Held (2005). *Gaussian Markov random fields: theory and applications*. CRC Press.

Sahu, S. K., A. A. Zhigljavsky, et al. (2003). Self-regenerative Markov chain Monte Carlo with adaptation. *Bernoulli 9*(3), 395–422.

Straatsma, T., H. Berendsen, and A. Stam (1986). Estimation of statistical errors in molecular simulation calculations. *Molecular Physics 57*(1), 89–95.

Thompson, M. B. (2010). Graphical comparison of MCMC performance. *arXiv preprint arXiv:1011.4457*.

Tibbits, M. M., C. Groendyke, M. Haran, and J. C. Liechty (2014). Automated factor slice sampling. *Journal of Computational and Graphical Statistics 23*(2), 543–563.

Turek, D., P. de Valpine, C. J. Paciorek, and C. Anderson-Bergman (2016). Automated parameter blocking for efficient markov chain Monte Carlo sampling. *Bayesian Analysis*.

Waller, L. A. and D. Zelterman (1997). Log-linear modeling with the negative multinomial distribution. *Biometrics*, 971–982.

Zipunnikov, V. V. and J. G. Booth (2006). Monte Carlo EM for generalized linear mixed models using randomized spherical radial integration.

Acknowledgement

This is funded in part by NSF SI2-SSI grant (ACI-1550488) and NSF Collaborative Research grant (DMS-1622444).

A Block MCMC sampling

This section provides the validity of block sampling in our general framework. We apply the results of Łatuszyński et al. (2013) for block Gibbs samplers. Following closely their notation, let $(\mathcal{X},\ B(\mathcal{X}))$ be an m-dimensional state space where $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$, $\mathcal{X}_i \subseteq \mathbb{R}^{b_i}$ so that the total dimension is $m = b_1 + \cdots + b_d$. We write $X_n \in \mathcal{X}$ as $X_n = (X_{n,1},\ \ldots,\ X_{n,d})$ and set $\mathcal{X}_{-i} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_{i-1} \times \mathcal{X}_{i+1} \times \cdots \times \mathcal{X}_d$ so that

$$X_{n,-i} := (X_{n,1},\ \ldots,\ X_{n,i-1},\ X_{n,i+1},\ \ldots,\ X_{n,d}).$$

For $X \sim \pi$, let the conditional distribution of $X_i|X_{-i} = x_{-i} \sim \pi(\cdot|x_{-i})$. We now consider a class of adaptive block MCMC sampler algorithms as follows.

ALGORITHM of Auto Adapt Block MCMC.

(1) Set $(\iota_n, \theta_n) := R_n(\ X_{n-1},\ \ldots,\ X_0,\ \iota_{n-1}, \theta_{n-1},\ \ldots,\ \iota_0, \theta_0) \in \bar{\Theta}$ where R_n is an outer adaptation function to select (ι_n, θ_n) in $\bar{\Theta}$.

(2) Choose block $i \in \{1,\ \ldots,\ d\}$ in that order.

(3) Draw $Y \sim Q_{X_{n-1,-i}, \iota_{n-1,i}, \theta_{n-1,i}}(X_{n-1,i},\ \cdot)$ where Q is a proposal distribution.

(4) Set $\begin{cases} X_n = \ (X_{n-1,1},\ \ldots,\ X_{n-1,i-1},\ Y,\ X_{n-1,i+1},\ \ldots,\ X_{n-1,d}) & \text{with probability } p \\ X_n = X_{n-1} & \text{otherwise} \end{cases}$

where $p = \min(1, \dfrac{\pi(Y|X_{n-1,-i}) q_{X_{n-1,-i}, \iota_{n-1,i}, \theta_{n-1,i}}(Y, X_{n-1,i})}{\pi(X_{n-1}|X_{n-1,-i}) q_{X_{n-1,-i}, \iota_{n-1,i}, \theta_{n-1,i}}(X_{n-1,i}, Y)})$.

To establish the results for Auto Adapt Block MCMC, we need the following assumption.

Assumption 1. *For every $i \in \{1,\ \ldots,\ d\}$, $x_{-i} \in \mathcal{X}_{-i}$ and $(\iota_i, \theta_i) \in \bar{\Theta}$, the transition kernel $P_{x_{-i}, \iota_i, \theta_i}$ is uniformly ergodic. Moreover, there exist $s_i > 0$ and an integer m_i s.t. for every $x_{-i} \in \mathcal{X}_{-i}$ and $(\iota_i, \theta_i) \in \bar{\Theta}$, there exists a probability measure $\nu_{x_{-i}, \iota_i, \theta_i}$ on $(\mathcal{X}_i,\ B(\mathcal{X}_i))$, s.t. $P^{m_i}_{x_{-i}, \iota_i, \theta_i}(x_i,\ \cdot) \geq s_i \nu_{x_{-i}, \iota_i, \theta_i}(\cdot)$ for every $x_i \in \mathcal{X}_i$.*

We have the following results, similar to Theorem 4.19 of Łatuszyński et al. (2013).

Theorem 1. *Assume that Assumption 1 holds. Then Auto Adapt Block MCMC is ergodic.*

That is, $\|\pi_n(x_0) - \pi\|_{\text{TV}} \to 0$ *as* $n \to \infty$. *Moreover, if*

$$\sup_{x_0} \sup_{x \in \mathcal{X}} \|P_{x_{-i}, \iota_{i+1}, \theta_{i+1}}(x_i,\ \cdot) - P_{x_{-i}, \iota_i, \theta_i}(x_i,\ \cdot)\|_{\text{TV}} \to 0$$

in probability, then convergence of Auto Adapt Block MCMC is also uniform over all x_0, *that is,*

$$\sup_{x_0} \|\pi_n(x_0) - \pi\|_{\text{TV}} \to 0$$

as $n \to \infty$.

Proof. To prove, we check simultaneous uniform ergodicity and the diminishing adaptation property of the algorithm and apply the result from Theorem 1 of Rosenthal and Roberts (2007) to conclude. We proceed as in the proof of Łatuszyński et al. (2013), Theorem 4.10 to establish simultaneous uniform ergodicity. By Assumption 1 and Lemma 4.14 of Łatuszyński et al. (2013), every adaptive Metropolis transition kernel for the ith block, $P_{x_{-i}, \iota_n, \theta_n}$, has stationary distribution $\pi(\cdot|x_{-i})$ and is $\left(\left(\left\lfloor \frac{\log(s_i/4)}{\log(1-s_i)} \right\rfloor + 2\right) m_i,\ \frac{s_i^2}{8}\right)$-strongly uniformly ergodic. Since each block is sampled at least once by our construction and there is only a finite dimension, the probability of selecting each block is positive and bounded away from 0. In other word, there exists ϵ such that $p_i > \epsilon$. Observe also that the family of block MCMC (p), $p_i > \epsilon$, is $(m',\ s')$-strongly uniformly ergodic for some $(m',\ s')$ by Łatuszyński et al. (2013) Lemma 4.15. Hence, the family of adaptive block Metropolis-within-Gibbs samplers indexed by $\iota_n, \theta_n \in \bar{\Theta}$, is $(m_*,\ s_*)$-simultaneously strongly uniformly ergodic with some m_* and s_* given as in (Roberts et al., 1998). Therefore we have that simultaneous uniform ergodicity is satisfied. Diminishing adaptation follows from Proposition 1 and from our construction. Therefore we can conclude the result. $\square$