Parallel and Distributed MCMC via Shepherding Distributions

Arkabandhu Chowdhury
Rice University

Chris Jermaine
Rice Univeristy

Abstract

In this paper, we present a general algorithmic framework for developing easily parallelizable/distributable Markov Chain Monte Carlo (MCMC) algorithms. Our framework relies on the introduction of an auxiliary distribution called a *shepherding distribution* (SD) that is used to control several MCMC chains that run in parallel. The SD is an introduced prior on one or more key parameters (or hyperparameters) of the target distribution. The shepherded chains then collectively explore the space of samples, communicating via the shepherding distribution, to reach high likelihood regions faster. The method of SDs is simple, and it is often easy to develop a shepherded sampler for a particular problem. Other advantages include wide applicability–the method can easily be used to draw samples from discrete distributions, or distributions on the simplex. Further, the method is asymptotically correct, since the method of SDs trivially maintains detailed balance.

1 INTRODUCTION

It is desirable to develop MCMC algorithms that use parallelization or distribution to run more efficiently [Williamson et al., 2013, Corander et al., 2006, Nishihara et al., 2014, Xu et al., 2014, Ahn et al., 2014, Wang et al., 2015, Desjardins et al., 2010]. Here, "more efficiently" can mean reaching a high likelihood portion of the data space in a shorter wall-clock time (that is, having a shorter burn-in period). This is often the goal in applications in AI or Bayesian machine learning, where MCMC is commonly used as an optimization method for solving MAP estimation problems, rather than out of any great devotion to the Bayesian approach. As an alternative, "more efficiently" can mean exploring a density function efficiently, producing samples that represent the full diversity of the high likelihood portion of the function's domain.

Unfortunately, since MCMC algorithms rely on simulating a Markov chain—a sequential construct—they are challenging to parallelize. Methods for parallelizing MCMC generally fall into two camps: those that are *data parallel* and those that are *transition parallel*.

Data parallel algorithms are far more common in the literature. These algorithms are typically useful for data-intensive Bayesian machine learning applications, where a large data set can be partitioned across many CPUs/machines, and some computationally expensive part of each simulation step can run over the different data partitions independently and in parallel [Yuan et al., 2015, Newman et al., 2009, Smola and Narayanamurthy, 2010]. Other popular data parallel algorithms run multiple simulations locally and then combine the results in a final step [Neiswanger et al., 2013, Wang and Dunson, 2013, Minsker et al., 2014].

Transition parallel algorithms attempt to use parallelism to make each state transition more effective, rather than more efficient. Thus, the goal is to run fewer, more effective steps. Unfortunately, when the goal is transition parallelism (or the problem to be solved does not lend itself to data parallelism), there are only a few general-purpose methods available. One of the most common (and oldest) methods for achieving transition parallelism involves running a number of chains in parallel, some of which have flattened probability density functions [Geyer, 1992]. Samples from the target density function are obtained by checking the state of a designated, non-flattened chain. Periodically, in a special Metropolis step, the identities of the chains are swapped. In theory, this can allow for better mixing, as the flattened chains facilitate a more thorough exploration of the sample space. This method is often used in computational biology, where sampling problems are not data parallelizable [Altekar et al., 2004].

In this paper, we propose a general approach to parallel MCMC that can obtain speedups through either data or transition parallelism, which we call the method of *shepherding distributions* (SDs). The method of SDs relies on

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84.

the introduction of an auxiliary distribution (the SD) that is then used to control several MCMC chains that run in parallel with a primary chain, which in turn is designed so as to have a stationary distribution equivalent to the target distribution, $f(x)$. The shepherded chains independently explore the solution space, but they are linked by the SD (which is essentially a prior on one or more of the key parameters to $f(x)$) so that when one shepherded chain reaches a high-likelihood solution, it can "pull" the other chains towards the same solution.

The method of SDs is related to the various "embarrassingly parallel" MCMC algorithms [Neiswanger et al., 2013], in that a number of chains are run independently. But rather than trying to combine the results after the run—which is difficult or impossible in the general case and may result in an algorithm with weak correctness guarantees—the SD ensures the various chains influence one another throughout the sampling process. Further, the use of an SD (which is often a simple conjugate prior) bakes this coordination into the sampling process, so correctness is guaranteed via detailed balance. Aside from its good performance, a key benefit of the method of SDs is its simplicity relative to other, recent alternatives in the literature. Further, SDs are widely applicable—SDs can easily be used to draw samples from mixed discrete and continuous distributions, or constrained distributions such as those on the simplex. The method of SDs is asymptotically exact and applicable to both data parallel and transition parallel problems.

We show through a number of different examples how the method of SDs can be used to design MCMC algorithms for difficult, real-world problems, that tend to outperform other general transition parallelization techniques (parallel (MC)^3 [Geyer, 1992] and parallel tempering [Earl and Deem, 2005]). In fact, in each of the applications that we consider, the method of SDs can be used to design algorithms that reach a high-likelihood region of the sample space that a conventional MCMC algorithm struggles to find, providing evidence that even a shepherded MCMC algorithm running on a single CPU may be superior to a conventional, un-shepherded algorithm.

2 THE SHEPHERDING APPROACH

Assume we wish to develop an MCMC algorithm for an unnormalized target distribution $f(x)$ over some state space $\mathcal{X}$. The method of SDs maintains a Markov chain, referred to as the *primary* chain, so designed that its stationary density is precisely the target density $f(x)$.

Now, we choose one of the fixed parameters (or hyperparameters) θ used in the formulation of $f(x)$ and view it as a random variable, introducing a prior $g(\theta)$ on θ, which we call the *shepherding distribution*. We then define a *shepherded chain* to be a Markov chain designed to have a stationary distribution $f'(x \mid \theta)$.

Often, f' is chosen so that $f' \equiv f$, but this is not necessary. In our first example of the method of SDs (solving a weighted MAX-SAT problem) f' is constructed to have the same mode(s) as f, but to have a higher variance, so that at least initially, each shepherded chain is given more freedom to explore the space $\mathcal{X}$.

When using the method of SDs, we augment the chain whose stationary distribution is $f(x)$ with $(n-1)$ additional shepherded chains, as well as a sampler for the shepherding parameter θ. In addition, we introduce an auxiliary variable $\boldsymbol{c}$ that holds a permutation of the numbers $\{1, ..., n\}$; the value stored in c_1 is the identity of the primary chain. Given a particular value of $\boldsymbol{c}$, the primary chain (identified by c_1) runs independently, whereas the shepherded chains (identified by $c_2, ..., c_n$) are bound by the shepherding parameter θ, through $g(\theta)$. Then, assuming that the prior on $\boldsymbol{c}$ is uniform over all permutations, the entire Markov process is designed to have unnormalized stationary density

$$f(x_{c_1})g(\theta)\prod_{i=2}^{n} f'(x_{c_i} \mid \theta). \tag{1}$$

Note that the method of SDs does not prescribe the exact nature of the Markov chain simulation used to ensure the required stationary distribution. Any strategy that maintains detailed balance can be used (Gibbs sampling, Metropolis-Hastings, Hamiltonian MCMC [Neal et al., 2011], etc.). In fact, it may make sense to use different strategies for different chains. Often $\boldsymbol{c}$ is updated using a Metropolis step that attempts to exchange the value of c_1 with a randomly selected c_i with probability

$$\min(1, \pi_i)$$

where

$$\pi_i = \frac{f(x_{c_i})f'(x_{c_1}|\theta)}{f(x_{c_1})f'(x_{c_i}|\theta)}.$$

This allows a high-quality solution computed by the shepherded chains to be exchanged into the primary chain. Or, one can compute π_i for each value of $i \in \{1...n\}$ and then choose a swap of c_1 with a randomly selected c_i in a Gibbs sampling step, with probability proportional to π_i.

In the most general case, the various shepherded chains may sample from different distributions. One can construct a chain whose stationary distribution is

$$f_1(x_{c_1})g(\theta)\prod_{i=2}^{n} f_i(x_{c_i} \mid \theta) \tag{2}$$

with the goal of drawing samples from f_1. This is particularly useful if the task is to realize an algorithm achieving data parallelism, so that the samplers, for each f_i where $i \geq 2$, are operating over subsets of a large data set in a

distributed machine learning setting (see our application to LDA [Blei et al., 2003] later in the paper). In such a setting, the shepherding distribution $g(\theta)$ provides a way for the various chains to communicate with one another, even though they are operating over different subsets of the data.

Thus, the method of SDs simulates a Markov chain having three types of dynamics: a periodic update of the permutation $\boldsymbol{c}$ (a "swapping step"), interleaved with parallel simulation of the chains corresponding to each x_i, as well as periodic updates of the shepherding variable θ. An example, shepherded MCMC algorithm that maintains detailed balance via Gibbs sampling, with a Metropolis update of $\boldsymbol{c}$, is given as Algorithm 1.

Algorithm 1 Algorithm for the method of SDs

Initialize $\theta \sim g(\theta)$
Initialize n Markov chains, $x_1, ..., x_n$
Initialize $\boldsymbol{c} \leftarrow \text{randperm}(n)$
while not converged **do**

// Update the state of each chain
Sample $x_{c_1} \sim f_1(x_{c_1})$
for $i \in \{2 \cdots n\}$ **do in parallel**
Sample $x_{c_i} \sim f_i(x_{c_i} \mid \theta)$
end for

// Update the shepherding distribution
Sample $\theta \sim g(\theta) \prod_{i=2}^{n} f_i(x_{c_i} \mid \theta)$

// Update the identity of the primary chain
$i \leftarrow \text{randint}(2, n)$
$\pi \leftarrow \frac{f_1(x_{c_i}) f_i(x_{c_1}|\theta)}{f_1(x_{c_1}) f_i(x_{c_i}|\theta)}$
if $\text{rand}(1) < \min(1, \pi)$ **then**
swap (c_i, c_1)
end if
end while
Return x_{c_1} as a sample from f_1

Note that while all three types of dynamics are performed in each iteration of Algorithm 1, it will often be beneficial to skip one or more of these in each iteration, for example, focusing more computational resources on the parallel update of each x_i.

Discussion. At a high level, the method of SDs works by running a number of shepherded Markov chains in parallel. As the chains begin to converge towards a common, high-likelihood state, the variance permitted by the hyperparameter set θ reduces, directing all the chains toward that high-likelihood state. If the state of the chain is complex and consists of many variables—as is often the case in Bayesian machine learning—the high-likelihood chains may only agree on a subset of their variables. In this case, given an appropriate choice of shepherding distribution, variance will be reduced only for those variables, leaving the chains free to explore the state space for the rest. In this way, the method of SDs resembles simulated annealing [Van Laarhoven and Aarts, 1987], but the tempering happens naturally, without the need for an explicit temperature value to control the state of the system. Further, the method of SDs facilitates solutions that "cool" the system only with respect to those portions of the state space for which there is agreement across chains.

The method of SDs can also be seen as related to the classical method of parallelizing/distributing stochastic learning algorithms that has long been part of the folklore: run the algorithm independently at different sites, and then periodically average the learned model [Zinkevich et al., 2010] or at the end of the computation [Neiswanger et al., 2013]. However, while averaging only makes sense for certain types of problems (averaging discrete models is problematic, for example), the method of SDs can work with any type of data for which an appropriate shepherding prior can be chosen. And while averaging often has very weak correctness guarantees, replacing an average with a Bayesian update, where the shepherding parameter θ is sampled as a random variable with density proportional to $g(\theta) \prod_{i=2}^{n} f_i(x_{c_i}|\theta)$, maintains detailed balance, and hence correctness is guaranteed.

We note that the choice of exact shepherding mechanism is important, but in our experience is often obvious from the problem context. One typically chooses a shepherding distribution as a prior on the key variables of interest. This allows the various shepherded chains to communicate with one another, via the hyperparameter set θ learned cooperatively.

Roadmap. In the next three sections of the paper, we give three examples of the application of the method of SDs. Our first two examples (weighted MAX-SAT and Bayesian linear regression with a spike-and-slab prior) will focus on transition parallelism, where the goal is to use parallelism that requires fewer epochs to converge. Our last example will focus on using the method of SDs to develop a data parallel sampler.

3 WEIGHTED MAX-SAT

In this section, we consider a shepherded solution to the weighted MAX-SAT problem. We show that using the method of SDs, it is easily possible to produce an MCMC sampler that outperforms other popular parallel samplers as well as a Gibbs sampler on the same problem.

3.1 PROBLEM FORMULATION

Maximum satisfiability problem (MAX-SAT) is the problem of determining the maximum number of clauses of a given Boolean formula in conjunctive normal form, that can be made true by an assignment of truth values to the

variables of the formula. The weighted MAX-SAT problem is a MAX-SAT problem in which each clause is given a positive weight, and the objective is to maximize the sum of weights of satisfied clauses by any assignment.

An instance of weighted MAX-SAT can be viewed as inducing a probability distribution as follows. Let $\mathbf{L}$ be the set of all the literals present in a MAX-SAT problem instance, and l_i be the ith literal in $\mathbf{L}$. L is a subset of $\mathbf{L}$ containing only the literals assigned to true. T_j (the jth clause in the SAT formula) is a binary-valued function over an assignment L. For example, a particular clause may take the form:

$$T_j(L) = 1 \text{ if } l_2 \in L \text{ and } l_4 \notin L \text{ and } l_5 \in L \text{ and } l_8 \in L$$
$$0 \text{ otherwise}$$

(We will subsequently use $\mathbf{L}_j$ to denote the set of literals named in T_j).

Now consider a random variable x that gives a random assignment of truth values to each of the literals in $\mathbf{L}$. In a weighted MAX-SAT problem, let m be the number of clauses, and w_j be the weight associated with clause j. For some constant ρ, if we define the probability of x taking the value L as:

$$P(x = L) \propto \prod_{j=1}^{m} \exp(T_j(L) \times \rho \times w_j) \tag{3}$$

then solving the weighted MAX-SAT problem is equivalent to finding the value of L that maximizes the value of the PMF in Equation 3. Note that in this formulation, ρ has no effect on the identity of the assignment that maximizes the PMF. If one uses an MCMC-based sampler to "solve" the resulting maximization problem and finds that the majority of the probability mass is concentrated away from the value of L that maximizes the value of the PMF, in theory one may force a sampler to choose the most likely samples by using a larger value of ρ.

Given this formulation, a simple Gibbs sampler for generating a Markov chain simulation $\langle x^{(0)}, x^{(1)}, ... \rangle$, whose stationary distribution is as given in Equation 3, is given in Algorithm 2. This algorithm loops through each of the clauses. For each clause, it enumerates all possible assignments of the literals named by the clause, and chooses one with probability proportional to the quality of the solution obtained by incorporating the possible assignment into the current solution.

3.2 A SHEPHERDED ALGORITHM

In practice this algorithm may quickly climb to a highly likely solution—which may not be optimal—and become stuck there. We can easily develop a parallel algorithm that addresses this problem using the method of SDs by introducing a prior probability θ_l that $l \in \mathbf{L}$ appears in a sampled assignment. Each shepherded chain samples assignments from a modified PMF taking the prior assignment probabilities into account:

Algorithm 2 Gibbs sampler for MAX-SAT

```
Initialize x^(0) to be a subset of L
for k ∈ {1 ··· big} do
  x^(k) ← x^(k−1)
  for j ∈ {1 ··· m} do
    sample L′ from PowerSet(L_j),
    s.t. Pr[selecting L′] ∝ P(x = (x^(k) − L_j) ∪ L′)
    x^(k) ← (x^(k) − L_j) ∪ L′
  end for
end for
```

$$P'(x = L|\theta) \propto \prod_{l \in L} \theta_l \prod_{l \in \mathbf{L}-L} (1 - \theta_l) \times \prod_{j=1}^{m} \exp(T_j(L) \times \rho' \times w_j).$$

Here, ρ' is a special multiplier used with the shepherded chains. For the corresponding shepherded sampler, $f(x)$ in Equation 1 is realized via the function $P(x = L)$, while $f'(x|\theta)$ in Equation 1 is realized via $P'(x = L|\theta)$ above. Further, we use a Beta prior on each element of θ, so

$$g(\theta) = \prod_{l \in \mathbf{L}} \text{Beta}(\theta_l | 0.1, 0.1)$$

will be used to "shepherd" various chains.

In a parallel implementation of our shepherded sampler, each shepherded chain runs one complete epoch of the Gibbs sampler (considering each of the clauses in turn) followed by a global (and generally inexpensive) update of θ. To give the shepherded chains the flexibility to explore the solution space, a relatively small value for the multiplier ρ' can be chosen; we use $\rho' = 0.01$. In a simple Gibbs sampler, using such a small value for ρ would be problematic as the chain would not explore the modes of the distribution, but this tends not to be a problem for shepherded chains. As a consensus is reached, the strength of the prior tends to pull the various chains into the mode of the distribution.

3.3 EVALUATION

We performed some empirical comparisons of our shepherded MCMC with the Gibbs sampler described previously as well as two methods for parallelizng MCMC algorithms: Parallel Metropolis-Coupled MCMC (p(MC)^3) [Altekar et al., 2004] and Hybrid Parallel Tempering Simulated Annealing (hPT/SA) [Li et al., 2009].

Parallel (MC)^3 is a parallelized Metropolis-Coupled MCMC, where a pool of chains heated to different temperatures are run in parallel (in the case of weighted MAX-SAT, the temperature corresponds to the value of ρ; the ith

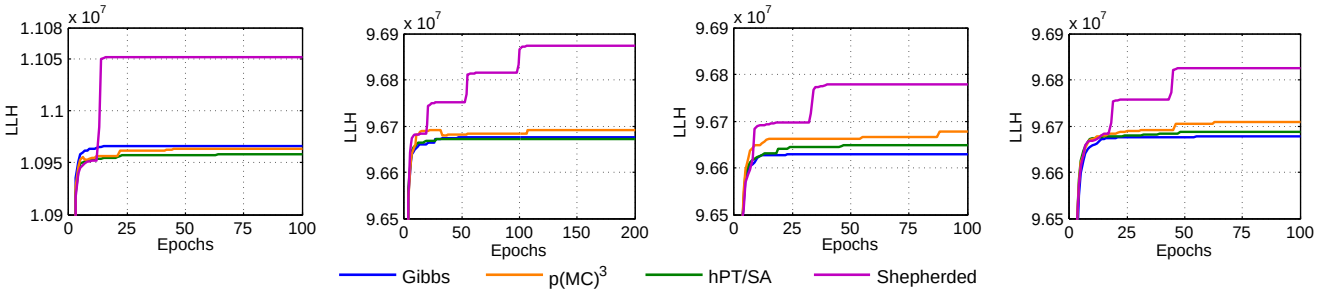

Figure 1: LLH of various MCMC samplers for four different Weighted MAX-SAT problems.

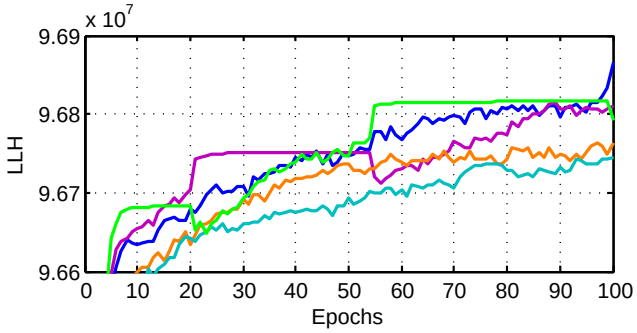

Figure 2: LLH for the five chains used in a Shepherded MCMC sampler to solve the second weighted MAX-SAT problem.

chain has its own ρ_i value, where $0 \leq \rho_i \leq 1$). The various chains form a 'temperature ladder' from cold to hot. The chains with $\rho_i < 1$ are called heated chains, allowing transitions out of local maxima more easily compared to the cold chain (with $\rho_1 = 1$). Periodically, in a Metropolis step, the states of various chains are exchanged with each other. This allows a hotter chain that has moved out of a local optimal to exchange its state with a cooler chain that has settled into such an optimal.

Hybrid PT/SA is a hybrid of Parallel Tempering (PT) [Earl and Deem, 2005] and Simulated Annealing (SA) [Van Laarhoven and Aarts, 1987]. Again, multiple chains at different temperature levels are run in parallel, where each chain makes a local update and every pair of neighboring chains in the temperature ladder attempt to swap states. At the same time, a SA is performed, where the temperature at each temperature level is gradually reduced, and thus the overall composite system is gradually guided towards the target temperature, while always remaining in a state close to thermodynamic equilibrium.

All of the parallel algorithms are implemented using five threads, and correspondingly five Markov chains. We evaluated all four algorithms on several popular weighted MAX-SAT benchmark problems[1], generating samples from the distribution induced using $\rho = 1$ [Hyttinen et al., 2014]. The problems are of moderate to large size. The number of literals varies from 12,764 to 40,290 over the four problems we tackle in this paper, and the number of clauses ranges from 46,236 to 145,910. In Figure 1, we show the comparison plots of the average log-likelihood (LLH) over ten independent runs of all four samplers. Rather than relying on wall-clock time, the x axis of each plot is the number of epochs of each sampler run. An epoch for the Gibbs sampler corresponds to a cycle through each of the clauses in the MAX-SAT problem. An epoch for the other three algorithms consists of an update for each of the clauses for each individual chain, as well as whatever attempts at swapping were required, or update to the shepherding distribution. Our rationale for considering epochs rather than wall-clock time is a desire to avoid measuring implementation effects (including the quality of the parallel implementation), and since the computational effort required to cycle through each clause should dominate, using the number of cycles for each chain as the x-axis seems to make the most sense.

Discussion. In each case, the SD-based method reaches a significantly higher likelihood than the other methods. For example, in Figure 1-(b), the SD-based method reaches an LLH as much as 200,000 higher than vanilla Gibbs sampler, and more than 150,000 higher than the other parallel algorithms (since $\rho = 1$, LLH is equivalent to the score of the resulting solution). To put this in perspective, the average clause weight is 150, so this additional LLH equates to a solution that includes around 1,000 more satisfied clauses than any other method.

It is interesting to note the step-like increase in LLH under the SD-based sampler. We can illustrate the reason for this in Figure 2, which shows the LLH for each of the chains corresponding to the five threads. Since the primary chain has a ρ value of one (whereas the shepherded chains use $\rho' = 0.01$) after a swap, the new primary chain sees a rapid increase in LLH as it adjusts to the higher ρ value, leaving the shepherded chains behind. However, the primary chain soon reaches a locally optimal solution from which it cannot escape. Then, as the shepherded chains collectively begin to agree with one another on certain literals, the corresponding prior hyperparameters reflect this agreement, and the shepherded chains show a gradual increase in LLH. Eventually, one of the shepherded chains achieves an

[1] www.cs.helsinki.fi/group/coreo/benchmarks/

LLH significantly higher than the primary chain, triggering a swap, a rapid rise in LLH, and the cycle begins again.

4 BAYESIAN LINEAR REGRESSION

We now consider a shepherded sampler for a high-dimensional Bayesian linear regression learning problem, with a spike-and-slab prior [Ishwaran and Rao, 2005] on the regression coefficients.

4.1 PROBLEM FORMULATION

We have a data set of m D-dimensional regressors $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m$. For a vector of regression coefficients $\mathbf{w}$,

$$y_i = \mathbf{x}_i^\intercal \mathbf{w} + \epsilon_i, \tag{4}$$

where $\epsilon_i \sim \text{Normal}(0, \sigma^2)$. We use a spike-and-slab prior for $\mathbf{w}$, which is standard in Bayesian learning for sparsifying such a model. The particular spike-and-slab variant we use has generative process:

$$\begin{aligned} \mathbf{w}' &\sim \text{Normal}(\vec{0}, I) \\ z_d &\sim \text{Bernoulli}(p) \text{ for each feature dimension } d \\ w_d &\leftarrow z_d w'_d. \end{aligned}$$

$\mathbf{z}$ is the censoring variable, and controls which regression coefficients are zero. Thus, each regression coefficient is zero with probability p, and otherwise has a Normal prior on it with variance 1.

Obtaining the posterior distribution over these variables typically requires computing and performing matrix operations over the $D \times D$ Gram matrix of $\mathbf{X}$. For high-dimensional data, this is impractical. To handle high-dimensional data, we can develop a blocked Gibbs sampler for this problem. First, partition the columns (features) of $\mathbf{X}$ into B subsets (or "blocks"), where the number of columns, D', is small enough that we can comfortably perform matrix operations over a $D' \times D'$ matrix. Then:

1. Let $\mathbf{X}_b$ denote $\mathbf{X}$ projected so that only the features in the bth block remain. Likewise, let $\bar{\mathbf{X}}_b$ denote $\mathbf{X}$ projected so as to remove features in the bth block.

2. Let $\mathbf{w}_b$ denote $\mathbf{w}$ projected so that only the weights for the features in the bth block remain ($\mathbf{w}'_b$ is defined similarly, for the uncensored version of $\mathbf{w}$). Likewise, let $\bar{\mathbf{w}}_b$ denote $\mathbf{w}$ projected so as to remove the features in the bth block ($\bar{\mathbf{w}}'_b$ defined similarly).

3. And $\mathbf{z}_b$ projects the list of masked regression coefficients onto the bth block, while $\bar{\mathbf{z}}_b$ projects the list of masked regression coefficients so as to remove the masks for the bth block.

Given this, Algorithm 3 gives a simple Gibbs sampler for BLR. In an epoch, for each block, the set of regression coefficients are updated, and then the various variables controlling the censored regression coefficients (the various $z_d^{(k)}$ values) are updated.

Algorithm 3 Gibbs sampler for BLR

Initialize each $\mathbf{w}_b'^{(0)}, \mathbf{z}_b^{(0)}$
for $k \in \{1...\text{big}\}$ **do**
  Set each $\mathbf{w}_b'^{(k)} \leftarrow \mathbf{w}_b'^{(k-1)}$
  Set each $\mathbf{z}_b^{(k)} \leftarrow \mathbf{z}_b^{(k-1)}$
  for $b \in \{1...B\}$ **do**
    $\mathbf{y}_b \leftarrow \mathbf{y} - (\bar{\mathbf{w}}_b^{(k)\intercal} \bar{\mathbf{X}}_b^{(k)\intercal})^\intercal$
    $\Sigma_b^{-1} \leftarrow I + \frac{1}{\sigma^2} \text{ gram}(\mathbf{X}_b) \text{ diag}(\mathbf{z}_b^{(k)})$
    $\mu_b \leftarrow \frac{1}{\sigma^2} \Sigma_b (\mathbf{X}_b \text{ diag}(\mathbf{z}_b^{(k)}))^\intercal \mathbf{y}_b$
    $\mathbf{w}_b'^{(k)} \sim \text{Normal}(\mu_b, \Sigma_b)$
  end for
  for each feature dimension d **do**
    Compute $p^{z_d^{(k)}} (1-p)^{1-z_d^{(k)}} \times$
    $\prod_j \text{Normal}(y_j | \mathbf{x}_j^\intercal \mathbf{w}, \sigma^2)$
    for $z_d^{(k)} \in \{0, 1\}$ and sample $z_d^{(k)}$
  end for
end for

4.2 SHEPHERDED BLR

To apply the method of SDs to this Gibbs sampler, we simply augment the spike-and-slab prior, by introducing a few shepherded hyperparameters. Specifically, we shepherd the uncensored list of coefficients $\mathbf{w}'$ via the introduction of new hyperparameters Λ and γ (so that $\mathbf{w}' \sim \text{Normal}(\Lambda, \text{diag}(\gamma))$ and we shepherd each z_d value (controlling whether or not the dth coefficient is censored) by adding a probability π_d such that $z_d \sim \text{Bernoulli}(\pi_d)$. Hence, the shepherding parameter set $\theta = \langle \Lambda, \gamma, \pi \rangle$ and an appropriately chosen shepherding distribution

$$g(\theta) = \text{Normal}(\Lambda | \vec{0}, I) \prod_d \text{InvGamma}(\gamma_d | 1, 1) \times \text{Beta}(\pi_d | \alpha, \beta).$$

In our parallel, shepherded sampler, each chain maintains its own $\mathbf{w}'$, $\mathbf{z}$ pair. During an epoch, each chain uses Algorithm 3 to move from state $k-1$ to state k. The only modification is that the shepherded chains must take into account the parameter set θ, so we use

$$\Sigma_b^{-1} \leftarrow \text{diag}(\gamma_b)^{-1} + \frac{1}{\sigma^2} \text{ gram}(\mathbf{X}_b \text{ diag}(\mathbf{z}_b^{(k)}))$$
$$\mu_b \leftarrow \Sigma_b (\text{diag}(\gamma_b)^{-1} \Lambda + \frac{1}{\sigma^2} (\mathbf{X}_b \text{ diag}(\mathbf{z}_b^{(k)}))^\intercal \mathbf{y}_b$$

and we replace p with π_d when sampling $z_d^{(k)}$. At the end of each epoch, we then update θ. This is computationally

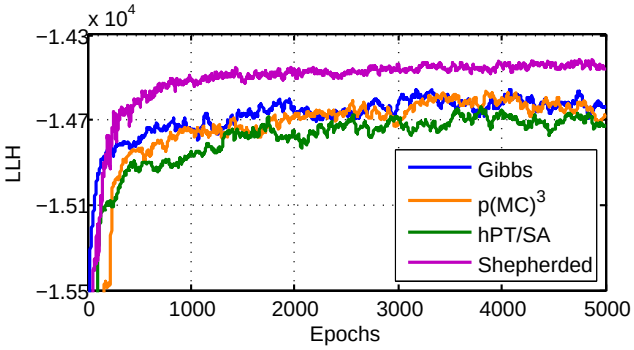

Figure 3: LLH of various samplers for BLR.

easy, due to conjugate priors, and much less expensive than updating each chain during the epoch.

4.3 EVALUATION

To test the utility of the shepherded algorithm, we use the BLR implementation to learn a model to determine whether or not a document from the 20 Newsgroups data set [Lang, 1995] is from one of the newsgroups related to religion, using the most frequent 2,000 words as our features. We again run three parallel MCMC implementations based upon the Gibbs sampler described in this section: p(MC)^3, hPT/SA, and the shepherded sampler, as well as a sequential Gibbs sampler. Each parallel MCMC implementation uses ten threads, and correspondingly ten Markov chains.

Figure 3 plots the average LLH over ten independent runs versus the number of epochs run.[2] Clearly, the shepherded sampler reaches a high likelihood much more quickly than the other three samplers. Further, after around 4,000 epochs, all four samplers seem to have stabilized in terms of the LLH achieved, with the shepherded sampler reaching a significantly higher likelihood than the other three samplers.

5 LATENT DIRICHLET ALLOCATION

We now consider a data parallel implementation of a sampler for Latent Dirichlet Allocation (LDA) [Blei et al., 2003]. Our focus is on examining samplers that are appropriate for a distributed environment where a subset of a large data set is stored on each compute node.

Basic Approach. Details of our shepherded LDA are given in the supplementary material. At a high level, assume that a compute cluster consists of $n-1$ machines; our shepherded algorithm will maintain $n-1$ shepherded chains and one primary chain. We partition the input data set D into $D_2 \cup D_3 \cup ... \cup D_n$ and locate each D_i on a different machine. The machine holding D_i will be responsible for maintaining the shepherded chain tasked with sampling x_{c_i}, using only the data in D_i. Since x_{c_1} corresponds to the primary chain, there is no D_1 to be operated on by the chain sampling x_{c_1}. In the case of data-parallel LDA, a sampled x_{c_i} consists of a complete set of model parameters, maintained locally, learned only over D_i.

In addition, the primary chain corresponding to x_{c_1} will be maintained in parallel by all machines, using a standard distributed LDA algorithm.

In LDA, the jth topic ψ_j is a set of word probabilities sampled from a Dirichlet(β) prior. The chains operating over the various D_i's are shepherded via the β hyper-parameter. That is, topic j is assigned its own hyper-parameter β_j. Periodically, the various chains communicate the set of statistics necessary to update β. When the chains begin to agree that word w is important (or unimportant) to topic j, then $\beta_{j,w}$ will be raised (or lowered) accordingly.

Evaluation. To evaluate the shepherded LDA algorithm, we consider the case where we have distributed an LDA computation over a cluster of ten machines. We consider two options for running a distributed LDA computation. In the first, we run the distributed Gibbs sampler described in this section. In the second, we run the shepherded sampler using eleven Markov chains (ten shepherded and one primary). As before, we measure the average LLH achieved over ten independent runs by both the samplers as a function of the amount of computation performed.

We ran the two samplers on two different data sets: the 20 Newsgroups data[3], and a corpus of Wikipedia articles[4]. For both data sets, we used 100 topics and a dictionary size of 10,000 words.

The results are plotted in Figure 4.[5] What is particularly interesting is that in both cases, the shepherded sampler reaches a significantly higher LLH than the vanilla Gibbs sampler, though both samplers seem to have converged (burned in) after about 200 distributed aggregations on each of the two data sets. This is in-keeping with the rest of the experimental findings in the paper, where the shepherded samplers are typically able to reach significantly higher likelihoods than the non-shepherded samplers. It is also notable that the LLH plot is significantly less smooth for the shepherded sampler. This is the result of continuous swaps

[2]Note that we plot LLH as a function of time, rather than a direct measure of model prediction accuracy. We argue that since MCMC is optimizing for LLH, this is the correct comparison metric. Measuring the model itself may introduce effects due to model selection and hyperparamter choice.

[3]kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html

[4]dumps.wikimedia.org/enwiki/

[5]Note that we plot LLH as a function of time, rather than the more traditional perplexity. Similar to in BLR, we argue that since MCMC is optimizing for LLH, this is the correct comparison metric. In contrast, perplexity measures model quality, which is known to be sensitive to hyperparaeters, model size, and so on [Asuncion et al., 2009].

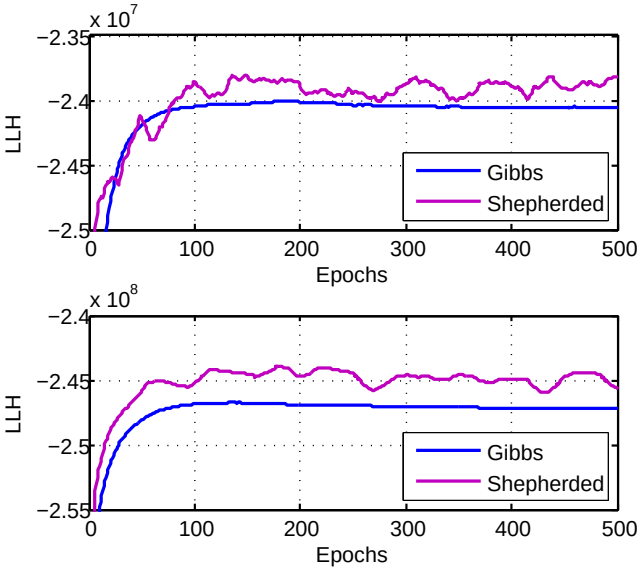

Figure 4: LLH for distributed Gibbs sampling and shepherded MCMC on the 20 Newsgroups data (top) and Wikipedia data (bottom).

for both data sets, which can result in either a precipitous drop or increase in LLH. We found that many more swaps happened when learning from the 20 Newsgroups data as opposed to the Wikipedia data.

6 RELATED WORK

There is relatively little work on developing general-purpose transition parallel MCMC algorithms. Most of the efforts are related to the idea of running multiple chains at different temperatures, which can be swapped periodically ($(MC)^3$ method [Altekar et al., 2004, Geyer, 1992]). Another way to manage transition parallelism is to use multiple cores to increase the speed of an individual step of an MCMC algorithm [Kontoghiorghes, 2005, Brockwell, 2006]. Generally these methods speed-up complex likelihood calculations by leveraging the conditional independence.

There has been a lot more research interest in developing frameworks that can be used to develop distributed MCMC algorithms, where the goal is to learn over a large data set partitioned over a compute cluster.

One very recent effort is distributed stochastic gradient MCMC [Ahn et al., 2014]. This algorithm is a variant of stochastic gradient MCMC [Ma et al., 2015], where the idea is to partition a large data set across machines, and then perform Bayesian inference using a stochastic gradient MCMC that jumps from machine to machine. The authors also propose running multiple such "jumping" samplers in parallel, and combining their results.

The most commonly appearing class of distributed MCMC algorithms in the literature is the set of so-called "embarrassingly parallel" MCMC algorithms. These distributed algorithms solve Bayesian learning problems by distributing data across a cluster, and then running independent chains on each subset of the data. At a later stage, the local samples are recombined into an approximation of samples from the desired global posterior of the entire data set [Neiswanger et al., 2013, Wang and Dunson, 2013, Minsker et al., 2014]. The key difficulty of such methods is approximating the global posterior from the individual posterior. There have been various distributed posterior approximation techniques used in literature, for example, Gaussian or Gaussian kernel density estimation (KDE) [Neiswanger et al., 2013], a Weierstrass transform representation of KDE [Wang and Dunson, 2013], and a median posterior in a reproducing kernel Hilbert space (RKHS) [Minsker et al., 2014], and recombining using random partition trees [Wang et al., 2015]. The main drawback of these samplings is that the local posteriors can significantly differ from each other due to noisy data, non-random partitioning of data, or due to simply not following the Gaussian assumptions in the final approximation, which can result in highly inaccurate global posterior samples. To overcome the significant difference among local posteriors, one idea is to use Expectation Propagation (EP) to facilitate sharing of moment statistics of local posteriors across nodes [Xu et al., 2014].

7 CONCLUSIONS AND DISCUSSION

The development of frameworks and algorithms for distributed MCMC is an important problem, and the method of SDs has several important advantages compared to other methods. SDs provide a simple and widely applicable method for parallelizing or distributing MCMC algorithms. SDs can handle arbitrary data types; any data for which a suitable prior can be found, may be shepherded. Further, the method of SDs is asymptotically exact (since SDs maintain detailed balance), and they are applicable to both data parallel and transition parallel problems. There are many opportunities for future research. One of the biggest problems with the method as described in this paper is the need for a primary chain in addition to the shepherded chains, in order to maintain asymptotic correctness. Checking for swaps between these chains is expensive, typically requiring a pass through the data in applications to machine learning. It is desirable to develop shepherded algorithms where the shepherded chains themselves are already guaranteed to sample from the target distribution, without the need for a primary chain.

Acknowledgements

Material in this paper was supported by the US National Science Foundation under grant number 1355998.

References

S. Ahn, B. Shahbaba, M. Welling, et al. Distributed stochastic gradient mcmc. In *ICML*, pages 1044–1052, 2014.

G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinformatics*, 20(3):407–415, 2004.

A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *UAI*, pages 27–34. AUAI Press, 2009.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3(Jan):993–1022, 2003.

A. E. Brockwell. Parallel markov chain monte carlo simulation by pre-fetching. *JCGS*, 15(1):246–261, 2006.

J. Corander, M. Gyllenberg, and T. Koski. Bayesian model learning based on a parallel mcmc strategy. *Statistics and computing*, 16(4):355–362, 2006.

G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau. Parallel tempering for training of restricted boltzmann machines. In *AISTATS*, pages 145–152. MIT Press Cambridge, MA, 2010.

D. J. Earl and M. W. Deem. Parallel tempering: Theory, applications, and new perspectives. *PCCP*, 7(23):3910–3916, 2005.

C. J. Geyer. Markov chain monte carlo maximum likelihood. Technical report, Minnesota Univ Minneapolis School Of Statistics, 1992.

A. Hyttinen, F. Eberhardt, and M. Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In *UAI*, pages 340–349, 2014.

H. Ishwaran and J. S. Rao. Spike and slab variable selection: frequentist and bayesian strategies. *Annals of Statistics*, pages 730–773, 2005.

E. J. Kontoghiorghes. *Handbook of parallel computing and statistics*. CRC Press, 2005.

K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, pages 331–339, 1995.

Y. Li, V. A. Protopopescu, N. Arnold, X. Zhang, and A. Gorin. Hybrid parallel tempering and simulated annealing method. *Applied Mathematics and Computation*, 212(1):216–228, 2009.

Y.-A. Ma, T. Chen, and E. Fox. A complete recipe for stochastic gradient mcmc. In *NIPS*, pages 2917–2925, 2015.

S. Minsker, S. Srivastava, L. Lin, and D. B. Dunson. Scalable and robust bayesian inference via the median posterior. In *ICML*, pages 1656–1664, 2014.

R. M. Neal et al. Mcmc using hamiltonian dynamics. *Handbook of MCMC*, 2:113–162, 2011.

W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*, 2013.

D. Newman, A. Asuncion, P. Smyth, and M. Welling. Doistributed algorithms for topic models. *JMLR*, 10 (Aug):1801–1828, 2009.

R. Nishihara, I. Murray, and R. P. Adams. Parallel mcmc with generalized elliptical slice sampling. *JMLR*, 15(1): 2087–2112, 2014.

A. Smola and S. Narayanamurthy. An architecture for parallel topic models. *VLDB*, 3(1-2):703–710, 2010.

P. J. Van Laarhoven and E. H. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, pages 7–15. Springer, 1987.

X. Wang and D. B. Dunson. Parallelizing mcmc via weierstrass sampler. *arXiv preprint arXiv:1312.4605*, 2013.

X. Wang, F. Guo, K. A. Heller, and D. B. Dunson. Parallelizing mcmc with random partition trees. In *NIPS*, pages 451–459, 2015.

S. Williamson, A. Dubey, and E. P. Xing. Parallel markov chain monte carlo for nonparametric mixture models. In *ICML (1)*, pages 98–106, 2013.

M. Xu, B. Lakshminarayanan, Y. W. Teh, J. Zhu, and B. Zhang. Distributed bayesian posterior sampling via moment sharing. In *NIPS*, pages 3356–3364, 2014.

J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma. Lightlda: Big topic models on modest computer clusters. In *WWW*, pages 1351–1361. ACM, 2015.

M. A. Zinkevich, A. Smola, M. Weimer, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, pages 2595–2603, 2010.