AN IMPROVED MERGE-SPLIT SAMPLER FOR CONJUGATE DIRICHLET PROCESS MIXTURE MODELS

David B. Dahl
dbdahl@stat.wisc.edu

Department of Statistics, and
Department of Biostatistics & Medical Informatics
University of Wisconsin – Madison

November 21, 2003

Technical Report #1086
Department of Statistics
University of Wisconsin – Madison

The author wishes to thank his Ph.D. adviser Michael Newton for helpful advice and Sonia Jain for access to data used in Jain and Neal (2004). The author is supported by the National Institutes of Health grant number EY07119. Software implementing conjugate Dirichlet process mixture models and the sampling algorithms presented in this paper are available at `http://www.stat.wisc.edu/~dbdahl/cdpmm/`.

AN IMPROVED MERGE-SPLIT SAMPLER FOR CONJUGATE DIRICHLET PROCESS MIXTURE MODELS

David B. Dahl

Abstract

The Gibbs sampler is the standard Markov chain Monte Carlo sampler for drawing samples from the posterior distribution of conjugate Dirichlet process mixture models. Researchers have noticed the Gibbs sampler's tendency to get stuck in local modes and, thus, poorly explore the posterior distribution. Jain and Neal (2004) proposed a merge-split sampler in which a naive random split is sweetened by a series of restricted Gibbs scans, where the number of Gibbs scans is a tuning parameter that must be supplied by the user.

In this work, I propose an alternative merge-split sampler borrowing ideas from sequential importance sampling. My sampler proposes splits by sequentially allocating observations to one of two split components using allocation probabilities that are conditional on previously allocated data. The algorithm does not require further sweetening and is, hence, computationally efficient. In addition, no tuning parameter needs to be chosen. While the conditional allocation of observations is similar to sequential importance sampling, the output from the sampler has the correct stationary distribution due to the use of the Metropolis-Hastings ratio.

The computational efficiency of my sequentially-allocated merge-split (SAMS) sampler is compared to Jain and Neal's sampler using various values for the tuning parameter. Comparisons are made in terms of autocorrelation times for four univariate summaries of the Markov chains taken at fixed time intervals. In four examples involving different models and datasets, I show that my merge-split sampler usually performs substantially better — in some cases, two to five times faster — than existing methods, and never performs worse.

Key Words: Conjugate Dirichlet process mixture model; Markov chain Monte Carlo; Metropolis-Hastings algorithm; split-merge updates; sequential importance sampling.

1 Introduction

Several computational strategies are available for fitting Dirichlet process mixture (DPM) models. These include sequential importance sampling (MacEachern, Clyde, and Liu 1999) and various Markov chain Monte Carlo (MCMC) algorithms. For a recent review, see MacEachern and Müller (2000) and (Neal 2000). Much of the literature focuses on fitting non-conjugate DPM models.

In the case of conjugate Dirichlet process mixture models, the Gibbs sampler is the most widely used MCMC algorithm. Unfortunately, the Gibbs samples updates the state space "one-at-a-time" and can get stuck in local modes and, thus, mix poorly across modes which have high probability.

To address the deficiencies of the Gibbs sampler, Jain and Neal (2004) propose a merge-split algorithm capable of making dramatic updates. By cycling between the Gibbs sampler and their sampler, they mitigate the mixing problem. Nevertheless, the computational effort required to fit the model can still be very substantial. The purpose of this paper is to introduce an alternative merge-split sampler which can greatly improve computational efficiency beyond that offered by Jain and Neal (2004).

Section 2 reviews the Dirichlet process mixture model and introduces the notation used to describe the sampling algorithms. In Sections 3.1 and 3.2, I describe the Gibbs sampler and the sampler of Jain and Neal (2004). Section 4 describes my alternative merge-split sampling algorithm. A comparison of these two merge-split samplers is given in Section 5. Concluding remarks are found in Section 6.

2 Dirichlet Process Mixture Model

The Dirichlet process mixture (DPM) model assumes that the observed data $\boldsymbol{y} = (y_1, \ldots, y_n)$ is generated from the following hierarchical model

$$\begin{aligned} y_i \mid \theta_i &\sim F(\theta_i) \\ \theta_i \mid G &\sim G \\ G &\sim \text{DP}(\alpha G_0), \end{aligned} \tag{1}$$

where $F(\theta_i)$ is a known parametric family of distributions indexed by θ and DP(αG_0) is the Dirichlet process (Ferguson 1973) centered about the distribution G_0 and having mass parameter $\alpha > 0$. The notation is meant to imply the obvious independence relations (e.g., y_1 given θ_1 is independent of the other y's, the other θ's, and G).

Blackwell and MacQueen (1973) show that $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ follows a general Polya urn scheme and may be represented in the following manner:

$$\begin{aligned} \theta_1 &\sim G_0 \\ \theta_i \mid \theta_1, \ldots, \theta_{i-1} &\sim \frac{\alpha G_0 + \sum_{j=1}^{i-1} \psi(\theta_j)}{\alpha + i - 1}, \end{aligned} \tag{2}$$

where $\psi(\theta)$ is the degenerate distribution at θ. Notice that (2) implies that $\theta_1, \ldots, \theta_n$ may share values in common, a fact that is used below in an alternative parameterization of $\boldsymbol{\theta}$. The model in (1) is simplified by integrating out the random mixing distribution G over its prior distribution in (2). Thus, the model in (1) becomes:

$$\begin{aligned} y_i \mid \theta_i &\sim F(\theta_i) \\ \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}) \text{ given in (2).} \end{aligned} \tag{3}$$

An alternative parameterization of $\boldsymbol{\theta}$ is given in terms of a set partition $\boldsymbol{\eta} = \{S_1, \ldots, S_q\}$ for $\{1, \ldots, n\}$ and a vector of component locations $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_q\}$, where $\phi_1, \ldots, \phi_q$ are paired with $S_1, \ldots, S_q$, respectively. A set partition $\boldsymbol{\eta}$ for $\{1, \ldots, n\}$ is a set of components (i.e., subsets) $S_1, \ldots, S_q$ such that $\cup_{i=1}^{q} S_i = \{1, 2, \ldots, n\}$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and $S_i \neq \emptyset$ for all i. One can now express (2) in terms of a prior on $\boldsymbol{\eta}$ and a conditional prior on $\boldsymbol{\phi}$ given $\boldsymbol{\eta}$. Equation (2) implies that the partition prior $p(\boldsymbol{\eta})$ can be written as:

$$\begin{aligned} p(\boldsymbol{\eta}) &= \alpha^q \prod_{j=1}^{q} \Gamma(|S_j|) \Big/ \prod_{i=1}^{n} (\alpha + i - 1) \\ &\propto \prod_{j=1}^{q} \alpha \Gamma(|S_j|), \end{aligned} \tag{4}$$

where $|S|$ is the number of elements of the component S and $\Gamma(x)$ is the gamma function evaluated at x. The specification of the prior under this alternative parameterization is completed by noting that $\phi_1, \ldots, \phi_q$ are independently drawn from G_0. Thus, $\boldsymbol{\theta}$ is equivalent to $(\boldsymbol{\eta}, \boldsymbol{\phi})$ and the model in

(1) and (3) may be expressed as

$$\begin{aligned} \boldsymbol{y} \mid \boldsymbol{\eta}, \boldsymbol{\phi} &\sim \prod_{i=1}^{n} F(\phi_1 \mathrm{I}\{i \in S_1\} + \ldots + \phi_q \mathrm{I}\{i \in S_q\}) \\ \boldsymbol{\phi} \mid \boldsymbol{\eta} &\sim \prod_{j=1}^{|\boldsymbol{\eta}|} G_0 \\ \boldsymbol{\eta} &\sim p(\boldsymbol{\eta}) \text{ given in (4)}, \end{aligned} \tag{5}$$

where $\mathrm{I}\{A\}$ is the indicator function for event A.

The partition likelihood $p(\boldsymbol{y}|\boldsymbol{\eta})$ is given as a product over components in $\boldsymbol{\eta} = \{S_1, \ldots, S_q\}$:

$$p(\boldsymbol{y}|\boldsymbol{\eta}) = \prod_{j=1}^{q} p(\boldsymbol{y}_{S_j}) \tag{6}$$

where

$$p(\boldsymbol{y}_S) = \int \prod_{k \in S} F(y_k; \phi)\, dG_0(\phi) \tag{7}$$

$$= \prod_{k \in S} \int F(y_k; \phi)\, dH_{<k}(\phi), \tag{8}$$

where $H_{<k}$ is the posterior distribution of a component location ϕ based on the prior G_0 and the data preceding the index k in the product above.

Combining the partition likelihood and the partition prior, Bayes rule gives the partition posterior as

$$p(\boldsymbol{\eta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\eta})\, p(\boldsymbol{\eta}), \tag{9}$$

where $p(\boldsymbol{y}|\boldsymbol{\eta})$ and $p(\boldsymbol{\eta})$ are given in (6) and (4), respectively.

If G_0 is conjugate to $F(\theta)$ in θ, the integral in (8) may be evaluated analytically. As a result, $p(\boldsymbol{\eta}|\boldsymbol{y})$ in (9) is available up to a normalizing constant. This technique was first used by MacEachern (1994) for mixture of normals and by Neal (1992) for models of categorical data. This technique has been shown to greatly improve the efficiency of Gibbs sampling (MacEachern 1994) and sequential importance sampling (MacEachern, Clyde, and Liu 1999). The Markov chain Monte Carlo sampling algorithm that I propose in this paper relies on conjugacy. I am currently working on extensions to non-conjugate models.

3 Existing MCMC Samplers

Using the notation of Section 2, this section reviews two common sampling methods for conjugate Dirichlet process mixture models.

3.1 Gibbs Sampler

One scan of the Gibbs sampler successively reallocates one index at a time according to the full conditional distributions given in (10) and (11). Let $\boldsymbol{\eta}^{-i}$ equal the set partition $\boldsymbol{\eta}$ in which the index i has been removed. Thus, $\boldsymbol{\eta}^{-i}$ is a set partition of $\{1, 2, \ldots, i-1, i+1, \ldots, n\}$. In keeping with the definition of a set partition, if the removal of the index i would result in $\boldsymbol{\eta}^{-i}$ containing an empty set, this empty set is discarded. One scan of the Gibbs sampling procedure is described next.

Algorithm: Gibbs Sampler

- For i in $\{1, 2, \ldots, n\}$,
  - Remove the index i from the state $\boldsymbol{\eta}$ to obtain $\boldsymbol{\eta}^{-i}$.
  - Obtain the new state $\boldsymbol{\eta}$ by allocating the index i to one of the components in $\boldsymbol{\eta}^{-i}$ or to the singleton set (with is then added to the state) with probability given by

$$\Pr(\, i \in S \mid \boldsymbol{\eta}^{-i}, \boldsymbol{y}\,) = b\,|S| \int F(y_i;\phi)\, dH_S(\phi), \qquad \text{for each } S \in \boldsymbol{\eta}^{-i}, \text{ and} \tag{10}$$

$$\Pr(\, i \in \{i\} \mid \boldsymbol{\eta}^{-i}, \boldsymbol{y}\,) = b\,\alpha \int F(y_i;\phi)\, dG_0(\phi), \qquad \text{otherwise,} \tag{11}$$

    where H_S is the posterior distribution of a component location ϕ based on the prior G_0 and the data corresponding to the indices in S. Also, b is a normalizing constant insuring the probabilities above sum to 1.
- The Metropolis-Hastings ratio will always be 1 and, therefore, every Gibbs scan is accepted.

Although the Gibbs sampler is very easy to implement, several authors have noted its deficiencies. Celeux, Hurn, and Robert (2000) state,

> The main defect of the Gibbs sampler from our perspective is the ultimate attraction of the local modes; that is, the almost impossible simultaneous reallocation of a group of observations to a different component . . . In many cases, the Gibbs sampler is unable to move the Markov chain to another mode of equal importance because of its inability to step over valleys of low probability. In addition, there is not way that one can judge whether or not the neighborhood of a specific mode has be sufficiently explored . . .

3.2 Restricted Gibbs Merge-Split Sampler

Jain and Neal (2004) propose a merge-split sampler for conjugate Dirichlet process mixture models. Being a merge-split algorithm, their method updates groups of indices in one update and thus is able to "step over valleys of low probability" and move between high-probability modes.

In this paper, the sampler of Jain and Neal (2004) is referred to as the restricted Gibbs merge-split sampler with t intermediate restricted Gibbs scans, abbreviated simply as the RGMS(t) sampler. As the name implies, the RGMS(t) sampler involves a modified Gibbs sampler. Instead of proposing naive random splits of components which are unlikely to be supported by the model, the RGMS(t) proposes splits that are more probable by sweetening a naive split. The algorithm reallocates the indices involved in the split among two split components through as series of t, Gibbs-style updates. The actually algorithm is described below.

Algorithm: RGMS(t) Sampler

- Uniformly select a pair of distinct indices i and j.
- If i and j belong to the same component in $\boldsymbol{\eta}$, propose $\boldsymbol{\eta}^\star$ by attempting a split move:
  - For convenience, denote the common component containing indices i and j as S.
  - Remove indices i and j from S and form singleton sets $S_i = \{i\}$ and $S_j = \{j\}$.
  - Make a naive split by allocating each remaining index in S to either S_i or S_j with equal probability.
  - Perform t restricted Gibbs scans of the indices in S. A restricted Gibbs scan differs from a regular Gibbs scan in that:
    - Instead of considering all the indices $\{1, 2, \ldots, n\}$, only the indices in S — the

original companions of i and j — are considered, and

* When reallocating an index, it can only be placed in one of two components: S_i and S_j. An index cannot be placed any other components and singleton components cannot be formed.

– Perform one final restricted Gibbs scan, this time keeping careful track of the Gibbs sampling transition probabilities for use in computing the Metropolis-Hastings ratio.

– Let $\boldsymbol{\eta}^\star$ be the set partition after the $t+1$ restricted Gibbs scans.

– Compute the Metropolis-Hastings ratio and accept $\boldsymbol{\eta}^\star$ as the new state of $\boldsymbol{\eta}$ with probability given by this ratio. Although it requires careful bookkeeping, computing the Metropolis-Hastings ratio is not difficult. See Jain and Neal (2004) for details on its computation.

- Otherwise, i and j belong to different components in $\boldsymbol{\eta}$. Propose $\boldsymbol{\eta}^\star$ by attempting a merge move:

  – For convenience, let S_i and S_j denote the components in $\boldsymbol{\eta}$ containing i and j, respectively.

  – Form a merged component $S = S_i \cup S_j$.

  – Propose the following set partition: $\boldsymbol{\eta}^\star = \boldsymbol{\eta} \cup \{S\} \setminus \{S_i, S_j\}$. In words, $\boldsymbol{\eta}^\star$ differs from $\boldsymbol{\eta}$ in that the two components containing indices i and j are merged into one component.

  – Compute the Metropolis-Hastings ratio and accept $\boldsymbol{\eta}^\star$ as the current state $\boldsymbol{\eta}$ with probability given by this ratio. Again, see Jain and Neal (2004) for a discussion of how to compute the Metropolis-Hastings ratio.

Notice that the RGMS(t) sampler requires the specification of the number of intermediate restricted Gibbs scans to perform before the final restricted Gibbs scan. This tuning parameter can be a blessing in that it allows flexibility in tailoring the algorithm for a particular situation. On the other hand, a practitioner may have little intuition as to what good value might be. A large number of restricted scans makes the split proposal more reasonable and therefore more likely to be accepted. Nevertheless, the marginal benefit of another restricted scan is decreasing in the number of scans and each additional scan takes time which could be used for another update. In the experience of Jain and Neal (2004), a "small number of scans (say, 4 or 6) is the best compromise."

While the RGMS(t) sampling algorithm can theoretically stand alone in sampling from the posterior distribution, Jain and Neal (2004) show that convergence is greatly improved by combining the RGMS(t) sampler with the traditional Gibbs sampler. They propose cycling between the two samplers by attempting x RGMS(t) updates and performing y Gibbs scans, where x and y are chosen by the practitioner. As a result, Gibbs sampling fine tunes the current state while RGMS(t) moves among modes.

4 Sequentially-Allocated Merge-Split (SAMS) Sampler

Fitting the posterior distribution of Dirichlet process mixture models can be an extremely computational task. While the Gibbs sampler will asymptotically explore the entire posterior distribution, using it by itself is problematic in a world of finite computational resources. It is not only important that sampling algorithms be correct; they must also be efficient. Cycling between the RGMS(t) sampler and the Gibbs sampler described in the previous section goes a long way toward the goal of improved efficiency. In this section, an alternative merge-split sampler is introduced. When combined with the Gibbs sampler, it appears to be more efficient than cycling Gibbs with the RGMS(t) sampler. This increased efficiency allows one to consider models that might otherwise be impractical to fit. The RGMS(t) and SAMS samplers are compared in Section 5.

The algorithm for my sequentially-allocated split-merge sampler is below. Conceptually, my sampler proposes splits by sequentially allocating observations to one of two split components using allocation probabilities that are conditional on previously allocated data. The algorithm does not require further sweetening — through say restricted Gibbs scans — and is, hence, computationally efficient. Also, no tuning parameter needs to be chosen.

Algorithm: SAMS Sampler

- Uniformly select a pair of distinct indices i and j.
- If i and j belong to the same component in $\boldsymbol{\eta}$, propose $\boldsymbol{\eta}^\star$ by attempting a split move:
  - For convenience, denote the common component containing indices i and j as S.
  - Remove indices i and j from S and form singleton sets $S_i = \{i\}$ and $S_j = \{j\}$.
  - Letting k be successive values in a uniformly-selected permutation of the indices in S,

add k to S_i with probability

$$\Pr(\, k \in S_i \mid S_i, S_j, \boldsymbol{y}\,) = \frac{|S_i| \int F(y_k;\phi) dH_{S_i}(\phi)}{|S_i| \int F(y_k;\phi) dH_{S_i}(\phi) + |S_j| \int F(y_k;\phi) dH_{S_j}(\phi)}, \quad (12)$$

where H_S is the posterior distribution of a component location ϕ based on the prior G_0 and the data corresponding to the indices in S. Otherwise, add k to S_j.

- Note that, at each iteration above, either S_i or S_j gains an index resulting in $|S_i|$ or $|S_j|$ increasing by 1. Further, H_{S_i} and H_{S_j} evolve to account for each additional index.
- Compute the Metropolis-Hastings ratio and accept $\boldsymbol{\eta}^\star$ as the current state $\boldsymbol{\eta}$ with probability given by this ratio. The calculation of the Metropolis-Hastings ratio is discussed in Section 4.1.

- Otherwise, i and j belong to different components in $\boldsymbol{\eta}$. Propose $\boldsymbol{\eta}^\star$ by attempting a merge move:
  - For convenience, let S_i and S_j denote the components in $\boldsymbol{\eta}$ containing i and j, respectively.
  - Form a merged component $S = S_i \cup S_j$.
  - Propose the following set partition: $\boldsymbol{\eta}^\star = \boldsymbol{\eta} \cup \{S\} \setminus \{S_i, S_j\}$. In words, $\boldsymbol{\eta}^\star$ differs from $\boldsymbol{\eta}$ in that the two components containing indices i and j are merged into one component.
  - Compute the Metropolis-Hastings ratio and accept $\boldsymbol{\eta}^\star$ as the current state $\boldsymbol{\eta}$ with probability given by this ratio. Again, the calculation of the Metropolis-Hastings ratio is discussed in Section 4.1.

The reader might notice that the algorithm resembles sequential importance sampling. In both algorithms, indices are allocated one-at-a-time and previously allocated indices are used in helping to choose how to allocate the current index. Some ways in which the algorithms differ include:

- In sequential importance sampling, all partitions are accepted, although they are not distributed according to the posterior distribution. To account for this, sequential importance sampling uses an importance weight when computing Monte Carlo integrals.

- The SAMS sampler is a Markov chain Monte Carlo algorithm whose stationary distribution is the correct posterior. Proposals are only accepted according to the probability given by the Metropolis Hastings ratio.

In contrast to the RGMS(t) sampler, the SAMS sampler does not have a tuning parameter that must be specified by the practitioner, making its application more automatic. As with the RGMS(t) sampler, cycling between the SAMS sampler and Gibbs scans is recommended. In the next section, the computation of the Metropolis-Hastings acceptance probability for the SAMS sampler is discussed.

4.1 Metropolis-Hastings Acceptance Probability

The Metropolis-Hastings (MH) ratio, denoted as $a(\boldsymbol{\eta}^\star|\boldsymbol{\eta})$, gives the probability that a proposed state $\boldsymbol{\eta}^\star$ is accepted from the current state $\boldsymbol{\eta}$. The MH ratio for the SAMS sampling algorithm is given as:

$$a(\boldsymbol{\eta}^\star|\boldsymbol{\eta}) = \min\left[1, \frac{p(\boldsymbol{\eta}^\star|\boldsymbol{y})}{p(\boldsymbol{\eta}|\boldsymbol{y})}\frac{\Pr(\boldsymbol{\eta}|\boldsymbol{\eta}^\star)}{\Pr(\boldsymbol{\eta}^\star|\boldsymbol{\eta})}\right], \tag{13}$$

where $p(\boldsymbol{\eta}^\dagger|\boldsymbol{y})$ is the partition posterior distribution evaluated at $\boldsymbol{\eta}^\dagger$ and $\Pr(\boldsymbol{\eta}^\ddagger|\boldsymbol{\eta}^\dagger)$ is the probability of proposing $\boldsymbol{\eta}^\ddagger$ from the state $\boldsymbol{\eta}^\dagger$. In practice, only parts of the partition posterior distribution need to be evaluated since contributions from components not involved in the split or merge will cancel.

When the proposal $\boldsymbol{\eta}^\star$ is a split update, $\Pr(\boldsymbol{\eta}^\star|\boldsymbol{\eta})$ is merely the product of the probabilities in (12) associated with the chosen allocations. Since these two split components could only be merged in one way, $\Pr(\boldsymbol{\eta}|\boldsymbol{\eta}^\star) = 1$. Conversely, when the proposal $\boldsymbol{\eta}^\star$ is a merge update, $\Pr(\boldsymbol{\eta}^\star|\boldsymbol{\eta})$ is 1, but $\Pr(\boldsymbol{\eta}|\boldsymbol{\eta}^\star)$ is the product of the probabilities in (12) associated with the allocation choices that would need to be made to obtain the split partition $\boldsymbol{\eta}$, although no actually splitting is performed. It is critical that a random permutation of the indices be used when performing this imaginary split.

5 Comparison of Merge-Split Samplers

This section compares the computational efficiency of my proposed sequentially-allocated merge-split (SAMS) sampler to the efficiency of the RGMS(t) sampler proposed by Jain and Neal (2004). Four examples are provided. In the comparisons, several values for t are chosen which depend somewhat on the particular model and data under consideration.

5.1 Methodology

The computational efficiency of the samplers is measured in terms of the autocorrelation time (ACT) of several univariate summaries of the states of the Markov chain. The ACT is defined as one plus two times the sum of the autocorrelations at lags one to infinity. The ACT of a quantity gives the factor by which the sample size is effectively reduced when estimating its expectation. For the purposes of this paper, the ACT was estimated using $10 \log_{10}(N)$ lags, where N is the number of draws from the Markov chain after burn-in. For each of the problems below, the burn-in and the number of draws is given. The univariate summaries considered in this paper are:

- Number of components in the state,
- Size of the largest component in the state,
- Log of the partition posterior distribution, and
- Entropy of the state, where entropy is defined as:

$$H(\boldsymbol{\eta}) = -\sum_{i=1}^{q} \frac{|S_i|}{n} \log\Big(\frac{|S_i|}{n}\Big).$$

As mentioned in the presentation of the RGMS(t) and SAMS sampler, it is recommended that these samplers be used in conjunction with the Gibbs sampler. Making these various samplers comparable requires a bit of thought. To take an extreme case, consider comparing the RGMS(1000) sampler to the RGMS(3). Certainly the RGMS(1000) will propose better splits since it can perform 1,000 intermediate Gibbs scans, but these scans come at a cost. For the amount of CPU time used to perform one RGMS(1000) iteration, perhaps hundreds of RGMS(3) could have been performed. Hence, RGMS(3) is likely to be more efficient in terms of CPU time.

For the comparisons in this paper, either 25%, 50%, or 75% of the CPU time is spent on Gibbs sampler updates. The remaining time is spent on one of the split-merge samplers under consideration. Snapshots of the current state are taken at uniform, fixed time intervals, regardless of the number of scans or merge-split updates performed. The time between snapshots is given in the discussion of each example. Because the number of possible combinations of the univariate summaries and Gibbs sampling times, this section only provides the tables of the ACT for the number

of components, the size of the largest component, and the entropy when 50% of time is spent on Gibbs sampling. The other tables are in the appendix.

Finally, it should be noted that all computations were performed on a computer with an AMD Athlon 1600+ processor and 512 MB of RAM running Red Hat Linux 8.0 using custom code written in Java and executed using IBM's Java Runtime Environment, version 1.4.1. The code is available at `http://www.stat.wisc.edu/~dbdahl/cdpmm/`.

5.2 Example 1: Jain & Neal's Model and Simulated Dataset

This example uses the model and data from example 3 of Jain and Neal (2004). Their data is simulated from a Bernoulli-Beta model. See their paper for details. The time interval between snapshots in the Markov chain is 0.01 seconds. The first 1,000 draws were discarded and 9,000 draws from the posterior were used to compute the autocorrelation times. The following tables give the estimated autocorrelation time and Monte Carlo standard error for the SAMS sampler and RGMS(t) samplers for various t. Notice that the SAMS sampler is significantly faster than any of the six RGMS samplers under consideration. Also notice that the Monte Carlo standard error is smaller for the SAMS sampler, indicating that the SAMS sampler is consistently better.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	1.65	0.08
2	RGMS(3)	2.16	0.13
3	RGMS(1)	2.19	0.11
4	RGMS(5)	2.46	0.12
5	RGMS(7)	2.79	0.23
6	RGMS(10)	3.19	0.23

Table 1: Auto-correlation time for number of components when 50% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	3.71	0.15
2	RGMS(3)	5.04	0.19
3	RGMS(1)	5.41	0.27
4	RGMS(5)	6.09	0.31
5	RGMS(7)	6.95	0.42
6	RGMS(10)	8.64	0.52

Table 2: Auto-correlation time for size of largest component when 50% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	4.17	0.19
2	RGMS(3)	6.41	0.32
3	RGMS(1)	7.07	0.44
4	RGMS(5)	8.15	0.43
5	RGMS(7)	9.46	0.70
6	RGMS(10)	12.31	0.85

Table 3: Auto-correlation time for entropy when 50% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

5.3 Example 2: Another Bernoulli-Beta Simulated Dataset

The second example uses another simulated dataset from the Bernoulli-Beta model in example 3 of Jain and Neal (2004). This data was chosen because the Gibbs sampler mixes slowly between two modes in this example. Again, the time interval between snapshots in the Markov chain is 0.01 seconds. The first 1,000 draws were discarded and 9,000 draws from the posterior were used to compute the autocorrelation times. Notice that the SAMS sampler is at least two times more efficient than any of the RGMS samplers under consideration. Again, the standard error associated with the SAMS sampler is smaller, indicating a consistent improvement.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	8.06	0.31
2	RGMS(5)	16.02	0.42
3	RGMS(4)	16.24	0.42
4	RGMS(3)	16.30	0.47
5	RGMS(7)	18.77	0.53
6	RGMS(2)	19.85	0.43
7	RGMS(1)	29.89	0.74

Table 4: Auto-correlation time for number of components when 50% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	9.24	0.33
2	RGMS(4)	18.96	0.48
3	RGMS(5)	18.97	0.51
4	RGMS(3)	19.07	0.55
5	RGMS(7)	22.15	0.60
6	RGMS(2)	22.87	0.44
7	RGMS(1)	34.79	0.78

Table 5: Auto-correlation time for size of largest component when 50% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	9.97	0.35
2	RGMS(5)	20.13	0.52
3	RGMS(4)	20.16	0.51
4	RGMS(3)	20.45	0.57
5	RGMS(7)	23.47	0.62
6	RGMS(2)	24.57	0.44
7	RGMS(1)	37.54	0.77

Table 6: Auto-correlation time for entropy when 50% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

5.4 Example 3: Gene Expression Model & Dataset

The third example is from a DNA microarray experiment which studies the response of the mouse heart to oxidative stress for two age groups over five time points. For more details on the experiment, see Dahl (2003b), although the model used here differs slightly.

While the original dataset contained 22,690 probe sets, the dataset was reduced (for illustrative purposes) to the 500 probe sets with the smallest p-value in a one-way ANOVA. The time interval between snapshots in the Markov chain is 0.1 seconds. The first 5,000 draws were discarded and 5,000 draws from the posterior were used to compute the autocorrelation times.

The case for the SAMS sampler being more efficient than any RGMS sampler is weaker in this case. The Monte Carlo standard errors are too large in relation to how close the autocorrelation times are. Nevertheless, in the 12 tables related to this example (three given here and nine in the appendix), the SAMS sampler has the smallest ACT in six cases. Under the assumptions that all of the samplers perform equally well and that the tables are independent, the binomial probability that the SAMS sampler would have the smallest ACT in 6 tables out of 12 is 0.008. Further, among the various RGMS samplers, no candidate emerges as a consistent rival to the SAMS samplers. Thus, even with the large Monte Carlo standard errors, a case can be made that the SAMS sampler is slightly more efficient in this example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	19.89	0.33
2	RGMS(3)	20.18	0.37
3	RGMS(10)	20.62	0.39
4	RGMS(5)	20.63	0.39
5	RGMS(1)	20.64	0.37
6	RGMS(20)	21.36	0.46

Table 7: Auto-correlation time for number of components when 50% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(1)	12.08	0.23
2	RGMS(5)	12.28	0.21
3	***SAMS***	12.43	0.22
4	RGMS(10)	12.45	0.20
5	RGMS(3)	12.48	0.26
6	RGMS(20)	12.83	0.23

Table 8: Auto-correlation time for size of largest component when 50% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(3)	34.60	0.67
2	***SAMS***	35.11	0.60
3	RGMS(5)	35.28	0.72
4	RGMS(1)	35.55	0.67
5	RGMS(10)	36.10	0.70
6	RGMS(20)	37.23	0.85

Table 9: Auto-correlation time for entropy when 50% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

5.5 Example 4: Univariate Gaussian Model & Datasets

The last example comes from a Gaussian mixture of univariate Gaussian random variables. For details of the model, see Dahl (2003a). The time interval between snapshots in the Markov chain is 0.01 seconds. The first 1,000 draws were discarded and 9,000 draws from the posterior were used to compute the autocorrelation times. Notice that the SAMS sampler performs somewhere between two times and four times better than any of the RGMS samplers under consideration. Again, the standard error associated with the SAMS sampler is smaller, indicating a consistent improvement.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	14.92	0.76
2	RGMS(3)	32.77	1.68
3	RGMS(20)	33.06	1.68
4	RGMS(1)	33.48	1.54
5	RGMS(5)	35.26	1.13
6	RGMS(10)	35.57	1.22

Table 10: Auto-correlation time for number of components when 50% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	6.47	0.69
2	RGMS(1)	41.37	2.03
3	RGMS(3)	45.23	1.89
4	RGMS(5)	48.05	2.02
5	RGMS(10)	51.65	1.38
6	RGMS(20)	59.22	1.44

Table 11: Auto-correlation time for size of largest component when 50% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	8.60	0.91
2	RGMS(1)	49.10	1.89
3	RGMS(3)	52.60	1.79
4	RGMS(5)	54.83	1.86
5	RGMS(10)	57.08	1.25
6	RGMS(20)	61.38	1.05

Table 12: Auto-correlation time for entropy when 50% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

6 Conclusion

This paper introduces a new split-merge MCMC algorithm for conjugate Dirichlet process mixture models. Using ideas from sequential importance sampling, my sampler proposes splits by sequentially allocating observations to one of two split components using allocation probabilities that are conditional on previously allocated data. The algorithm is computationally efficient because splits are quickly proposed and do not need any further sweetening. In addition, no tuning parameter needs to be chosen and hence the application of the sampler is more automatic. While the conditional allocation of observations is similar to sequential importance sampling, the output from the sampler has the correct stationary distribution due to the use of the Metropolis-Hastings ratio.

An important area of future research is applying my sequentially-allocated split-merge sampler to non-conjugate Dirichlet process mixture models. I am currently working on these extensions.

References

Blackwell, D. and J. B. MacQueen (1973). Ferguson distributions via Polya urn schemes. *The Annals of Statistics 1*, 353–355.

Celeux, G., M. Hurn, and C. P. Robert (2000). Computational and inferential difficulties with mixture posterior distributions. *Journal of the American Statistical Association 95*(451), 957–970.

Dahl, D. B. (2003a). Modal clustering in a univariate class of product partition models. Technical Report 1085, Department of Statistics, University of Wisconsin - Madison.

Dahl, D. B. (2003b). Modeling differential gene expression using a dirichlet process mixture model. In *ASA Proceedings of the Joint Statistical Meetings*.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics 1*, 209–230.

Jain, S. and R. M. Neal (2004). A split-merge Markov chain Monte Carlo procedure for the Dirichlet Process mixture model. *Journal of Computational and Graphical Statistics 13*(1), in press.

MacEachern, S. and P. Müller (2000). Efficient MCMC schemes for robust model extensions using ecompassing Dirichlet process mixture models. In *Robust Bayesian analysis*, pp. 295–315.

MacEachern, S. N. (1994). Estimating normal means with a conjugate style Dirichlet process prior. *Communications in Statistics, Part B – Simulation and Computation 23*, 727–741.

MacEachern, S. N., M. Clyde, and J. S. Liu (1999). Sequential importance sampling for non-parametric Bayes models: The next generation. *The Canadian Journal of Statistics 27*, 251–267.

Neal, R. M. (1992). Bayesian mixture modeling. In *Maximum Entropy and Bayesian Methods: Proceedings of the 11th International Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, pp. 197–211.

Neal, R. M. (2000). Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics 9*, 249–265.

A Supplemental Tables

A.1 Example 1: Jain & Neal's Model and Simulated Dataset

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	5.39	0.16
2	RGMS(5)	5.43	0.13
3	RGMS(1)	5.65	0.14
4	RGMS(3)	5.72	0.19
5	RGMS(7)	5.77	0.20
6	RGMS(10)	5.90	0.16

Table 13: Auto-correlation time for log posterior when 50% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	1.59	0.05
2	RGMS(3)	2.08	0.21
3	RGMS(1)	2.28	0.13
4	RGMS(5)	2.29	0.14
5	RGMS(7)	2.49	0.16
6	RGMS(10)	2.72	0.13

Table 14: Auto-correlation time for number of components when 25% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	3.77	0.11
2	RGMS(3)	4.99	0.34
3	RGMS(1)	5.35	0.37
4	RGMS(5)	5.77	0.19
5	RGMS(7)	6.41	0.29
6	RGMS(10)	7.37	0.24

Table 15: Auto-correlation time for size of largest component when 25% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	7.83	0.27
2	RGMS(5)	8.21	0.21
3	RGMS(1)	8.32	0.26
4	RGMS(7)	8.41	0.24
5	RGMS(3)	8.51	0.26
6	RGMS(10)	8.98	0.28

Table 16: Auto-correlation time for log posterior when 25% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	2.83	0.13
2	RGMS(3)	4.59	0.48
3	RGMS(5)	5.73	0.28
4	RGMS(1)	5.79	0.48
5	RGMS(7)	6.62	0.46
6	RGMS(10)	8.15	0.41

Table 17: Auto-correlation time for entropy when 25% of the CPU time was spent on the Gibbs sampler to fit the Jain, Neal (2004) example.

A.2 Example 2: Another Bernoulli-Beta Simulated Dataset

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	13.10	0.46
2	RGMS(4)	27.10	0.72
3	RGMS(3)	27.45	0.61
4	RGMS(5)	28.44	0.54
5	RGMS(2)	30.32	0.64
6	RGMS(7)	30.65	0.72
7	RGMS(1)	42.71	0.68

Table 18: Auto-correlation time for number of components when 75% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	15.44	0.54
2	RGMS(4)	32.28	0.79
3	RGMS(3)	32.63	0.64
4	RGMS(5)	33.89	0.63
5	RGMS(2)	35.73	0.64
6	RGMS(7)	36.24	0.79
7	RGMS(1)	49.88	0.71

Table 19: Auto-correlation time for size of largest component when 75% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	1.69	0.06
2	RGMS(4)	3.12	0.11
3	RGMS(3)	3.14	0.09
4	RGMS(2)	3.28	0.12
5	RGMS(5)	3.53	0.17
6	RGMS(7)	3.69	0.19
7	RGMS(1)	4.30	0.14

Table 20: Auto-correlation time for log posterior when 75% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	16.88	0.56
2	RGMS(4)	34.37	0.76
3	RGMS(3)	34.78	0.64
4	RGMS(5)	36.01	0.62
5	RGMS(2)	38.18	0.63
6	RGMS(7)	38.59	0.82
7	RGMS(1)	53.17	0.69

Table 21: Auto-correlation time for entropy when 75% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	1.42	0.04
2	RGMS(3)	2.43	0.11
3	RGMS(4)	2.54	0.10
4	RGMS(2)	2.62	0.09
5	RGMS(5)	2.68	0.09
6	RGMS(7)	2.86	0.15
7	RGMS(1)	2.92	0.10

Table 22: Auto-correlation time for log posterior when 50% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	5.09	0.13
2	RGMS(4)	11.26	0.26
3	RGMS(5)	11.83	0.39
4	RGMS(3)	12.52	0.35
5	RGMS(7)	12.89	0.36
6	RGMS(2)	14.78	0.31
7	RGMS(1)	25.41	0.79

Table 23: Auto-correlation time for number of components when 25% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	5.71	0.16
2	RGMS(4)	12.99	0.30
3	RGMS(5)	13.84	0.45
4	RGMS(3)	14.25	0.44
5	RGMS(7)	15.10	0.40
6	RGMS(2)	16.84	0.34
7	RGMS(1)	29.43	0.83

Table 24: Auto-correlation time for size of largest component when 25% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	1.58	0.04
2	RGMS(2)	2.35	0.06
3	RGMS(3)	2.40	0.07
4	RGMS(4)	2.58	0.09
5	RGMS(5)	2.83	0.11
6	RGMS(7)	2.92	0.15
7	RGMS(1)	2.96	0.09

Table 25: Auto-correlation time for log posterior when 25% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	6.15	0.16
2	RGMS(4)	13.88	0.31
3	RGMS(5)	14.69	0.47
4	RGMS(3)	15.28	0.44
5	RGMS(7)	15.93	0.42
6	RGMS(2)	18.18	0.36
7	RGMS(1)	31.84	0.86

Table 26: Auto-correlation time for entropy when 25% of the CPU time was spent on the Gibbs sampler to fit the Bernoulli-Beta example.

A.3 Example 3: Gene Expression Model & Dataset

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	17.13	0.34
2	RGMS(5)	17.24	0.37
3	RGMS(1)	17.44	0.36
4	RGMS(3)	17.63	0.30
5	RGMS(10)	17.76	0.34
6	RGMS(20)	18.03	0.35

Table 27: Auto-correlation time for number of components when 75% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(3)	8.31	0.17
2	RGMS(1)	8.44	0.18
3	***SAMS***	8.46	0.15
4	RGMS(5)	8.77	0.18
5	RGMS(10)	8.87	0.51
6	RGMS(20)	9.39	0.43

Table 28: Auto-correlation time for size of largest component when 75% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	9.35	0.19
2	RGMS(10)	9.50	0.24
3	RGMS(3)	9.66	0.22
4	RGMS(5)	9.91	0.20
5	RGMS(20)	9.93	0.22
6	RGMS(1)	10.02	0.24

Table 29: Auto-correlation time for log posterior when 75% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	31.08	0.61
2	RGMS(1)	31.52	0.59
3	RGMS(5)	31.57	0.72
4	RGMS(3)	31.60	0.70
5	RGMS(10)	31.84	0.83
6	RGMS(20)	32.45	0.71

Table 30: Auto-correlation time for entropy when 75% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(5)	11.62	0.21
2	RGMS(10)	12.10	0.29
3	***SAMS***	12.10	0.27
4	RGMS(1)	12.44	0.28
5	RGMS(3)	12.46	0.25
6	RGMS(20)	12.63	0.29

Table 31: Auto-correlation time for log posterior when 50% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	25.68	0.39
2	RGMS(1)	27.33	0.43
3	RGMS(3)	27.86	0.56
4	RGMS(5)	27.94	0.47
5	RGMS(10)	28.24	0.49
6	RGMS(20)	28.78	0.58

Table 32: Auto-correlation time for number of components when 25% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(20)	22.02	0.30
2	RGMS(3)	22.04	0.38
3	***SAMS***	22.16	0.33
4	RGMS(10)	22.27	0.38
5	RGMS(1)	22.28	0.41
6	RGMS(5)	22.32	0.33

Table 33: Auto-correlation time for size of largest component when 25% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	RGMS(20)	17.14	0.37
2	RGMS(3)	17.93	0.32
3	***SAMS***	18.00	0.38
4	RGMS(1)	18.08	0.32
5	RGMS(5)	18.19	0.35
6	RGMS(10)	18.52	0.44

Table 34: Auto-correlation time for log posterior when 25% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	40.28	0.57
2	RGMS(1)	43.21	0.64
3	RGMS(3)	43.52	0.51
4	RGMS(10)	44.24	0.62
5	RGMS(5)	44.31	0.53
6	RGMS(20)	44.48	0.77

Table 35: Auto-correlation time for entropy when 25% of the CPU time was spent on the Gibbs sampler to fit the gene expression example.

A.4 Example 4: Univariate Gaussian Model & Datasets

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	15.41	1.03
2	RGMS(1)	27.12	1.29
3	RGMS(3)	28.29	1.08
4	RGMS(5)	28.75	1.41
5	RGMS(10)	29.98	1.48
6	RGMS(20)	31.08	1.61

Table 36: Auto-correlation time for number of components when 75% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	9.46	1.04
2	RGMS(1)	42.65	1.76
3	RGMS(3)	48.03	1.28
4	RGMS(5)	50.76	1.27
5	RGMS(10)	56.32	1.17
6	RGMS(20)	59.39	1.25

Table 37: Auto-correlation time for size of largest component when 75% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	12.18	1.16
2	RGMS(1)	48.95	1.37
3	RGMS(3)	52.33	1.27
4	RGMS(5)	55.16	0.98
5	RGMS(10)	58.77	0.83
6	RGMS(20)	61.38	0.59

Table 38: Auto-correlation time for log posterior when 75% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	12.03	1.15
2	RGMS(1)	49.10	1.48
3	RGMS(3)	52.71	1.28
4	RGMS(5)	55.34	1.02
5	RGMS(10)	58.80	0.85
6	RGMS(20)	61.36	0.61

Table 39: Auto-correlation time for entropy when 75% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	8.78	0.91
2	RGMS(1)	48.35	1.70
3	RGMS(3)	52.21	1.76
4	RGMS(5)	54.38	1.86
5	RGMS(10)	56.94	1.22
6	RGMS(20)	61.24	0.99

Table 40: Auto-correlation time for log posterior when 50% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	18.70	0.94
2	RGMS(5)	42.41	1.36
3	RGMS(1)	43.21	1.44
4	RGMS(10)	43.30	1.76
5	RGMS(3)	43.86	1.35
6	RGMS(20)	44.93	1.73

Table 41: Auto-correlation time for number of components when 25% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	6.21	1.23
2	RGMS(1)	41.10	2.88
3	RGMS(3)	43.91	2.85
4	RGMS(5)	48.81	2.65
5	RGMS(10)	52.17	2.23
6	RGMS(20)	60.14	1.64

Table 42: Auto-correlation time for size of largest component when 25% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	7.85	1.01
2	RGMS(1)	51.07	2.09
3	RGMS(3)	52.29	2.18
4	RGMS(5)	55.67	2.34
5	RGMS(10)	58.78	1.86
6	RGMS(20)	64.24	1.27

Table 43: Auto-correlation time for log posterior when 25% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.

		Monte Carlo	
Rank	Sampler	Mean	Std. Error
1	***SAMS***	7.62	0.97
2	RGMS(1)	51.47	2.41
3	RGMS(3)	53.37	2.35
4	RGMS(5)	56.28	2.38
5	RGMS(10)	58.66	1.95
6	RGMS(20)	64.37	1.31

Table 44: Auto-correlation time for entropy when 25% of the CPU time was spent on the Gibbs sampler to fit the univariate Gaussian example.