A permutation-augmented sampler for DP mixture models

Percy Liang PLIANG@CS.BERKELEY.EDU
University of California, Berkeley

Michael Jordan JORDAN@CS.BERKELEY.EDU
University of California, Berkeley

Ben Taskar TASKAR@CIS.UPENN.EDU
University of Pennsylvania

Abstract

We introduce a new inference algorithm for Dirichlet process mixture models. While Gibbs sampling and variational methods focus on local moves, the new algorithm makes more global moves. This is done by introducing a permutation of the data points as an auxiliary variable. The algorithm is a blocked sampler which alternates between sampling the clustering and sampling the permutation. The key to the efficiency of this approach is that it is possible to use dynamic programming to consider all exponentially many clusterings consistent with a given permutation. We also show that random projections can be used to effectively sample the permutation. The result is a stochastic hill-climbing algorithm that yields burn-in times significantly smaller than those of collapsed Gibbs sampling.

1. Introduction

Dirichlet process (DP) mixture models (Antoniak, 1974) have been usefully employed as a clustering methodology in a variety of applied areas such as bioinformatics (Xing et al., 2004), vision (Sudderth et al., 2006), and topic modeling (Teh et al., 2006). By treating the number of mixture components as random, DP mixtures provide an appealing nonparametric approach to mixture modeling in which the complexity of the model adapts to the complexity inherent in the data.

Posterior inference for DP mixtures is challenging, and a variety of inference algorithms have been specialized to the DP mixture setting, including samplers (Ishwaran & James, 2001; Escobar & West, 1995), variational approximations (Blei & Jordan, 2005; Kurihara et al., 2007), and other search algorithms (Daume, 2007). A difficulty with all of these algorithms is their tendency to get trapped in local optima. Variational methods converge to local optima, and while samplers are guaranteed to converge to the correct posterior in the limit, they sometimes suffer from extremely slow mixing. For example, the popular collapsed Gibbs sampler (Escobar & West, 1995) reassigns only one data point at a time. These local moves make it difficult to make large changes to the clustering without having to step through low probability clusterings. Various split-merge algorithms have been developed to address this issue and provide more global moves (Jain & Neal, 2000; Dahl, 2003a).

In this paper, we develop a novel methodology for posterior inference in DP mixtures. Rather than focusing on local moves, we develop a method which allows us to sample an entire clustering at once, a move even more global than that offered by split-merge. Our approach is an instance of the general class of augmentation samplers, a class which includes the Swendsen-Wang sampler (Swendsen & Wang, 1987) and others (Tanner & Wong, 1987; Liu & Wu, 1999). Specifically, we augment the DP mixture to include a permutation (ordering) of the data points. We then alternate between sampling permutations and sampling clusterings. The key to our approach is the following insight: given a fixed permutation, all exponentially many clusterings consistent with the permutation can be considered using dynamic programming.

In related work, Friedman and Koller (2000) have exploited permutations in Bayesian network structure sampling to yield tractable subproblems. The idea of using dynamic programming for clustering has been used for finding the MAP clustering of univari-

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

ate data (Dahl, 2003b). Dynamic programming can also be applied on a tree rather than a sequence (Heller & Ghahramani, 2005). An important distinction between these algorithms and ours is that our permutation-augmented sampler converges to the exact posterior over clusterings.

While the key to our work is the fact that the sampling of a clustering conditioned on a permutation can be done efficiently, it is also necessary to develop efficient methods for sampling the permutation given the clustering. We show that this can be done using random projections (Johnson & Lindenstrauss, 1984) to construct an effective stochastic hill-climbing algorithm.

2. DP mixture models

Given a measurable space Ω, a base distribution G_0 and a concentration parameter α_0, the *Dirichlet process*, $\mathrm{DP}(\alpha_0, G_0)$, is a distribution over distributions on Ω uniquely defined by the following property: $G \sim \mathrm{DP}(\alpha_0, G_0)$ if and only if

$$(G(A_1), \ldots, G(A_K)) \sim \mathrm{Dir}(\alpha_0 G_0(A_1), \ldots, \alpha_0 G_0(A_K))$$

for all measurable partitions $A_1, \ldots, A_K$ of Ω (Ferguson, 1973).

Draws G from a Dirichlet process turn out to be discrete with probability one, making the Dirichlet process suitable for mixture modeling (Antoniak, 1974). In this case, Ω is the space of parameters of mixture components. The data points $\mathbf{x} = (x_1, \ldots, x_n)$ are generated as follows: draw a parameter $\theta_i \sim G$ and then generate $x_i \sim F(\cdot; \theta_i)$, where F is the probability model associated with a mixture component (e.g., Gaussian, multinomial, etc.).

Data points are clustered by virtue of sharing identical values of the parameter θ_i. For the purpose of this paper, it will be convenient to focus on this clustering rather than the values of the parameters. A *clustering* $\mathbf{C}$ contains $|\mathbf{C}|$ clusters, each *cluster* $c \in \mathbf{C}$ being a subset of the indices $\{1, \ldots, n\}$. Collectively, the clusters partition the set of points. For example, $\mathbf{C} = \{\{2, 3\}, \{1, 5\}, \{4\}\}$ is a clustering of 5 data points into 3 clusters.

One computationally useful representation of the Dirichlet process is the Chinese restaurant process (CRP) (Pitman, 2002), which describes the induced clustering $\mathbf{C}$ when G is marginalized out. The CRP constructs the random clustering sequentially. Each data point x_i is placed in an existing cluster $c \in \mathbf{C}$ with probability proportional to its size $|c|$, and a new cluster is created with probability proportional to α_0. An important property of the CRP is that despite its sequential definition, the distribution on partitions that it induces is exchangeable; i.e., the probability across partitions is the same for all orderings of the data points.

2.1. Existing CRP-based sampling algorithms

The collapsed Gibbs sampler is based on exchangeability of the CRP. Indeed, since the data points are exchangeable, we can compute the conditionals needed by a Gibbs sampler by pretending that a given data point is the final data point (Escobar & West, 1995). The cluster assignment of point x_i is thus sampled according to the following probabilities:

$$p(i \in c) \propto |c \backslash \{i\}| \int F(x_i; \theta) G_0(d\theta \mid \mathbf{x}_{c \backslash \{i\}}),$$

for an existing cluster c and

$$p(i \in c_{\text{new}}) \propto \alpha_0 \int F(x_i; \theta) G_0(d\theta)$$

for a new cluster c_{new}. While the Gibbs sampler is very simple to implement, it can be slow to mix. Only one point can be reassigned at a time, and thus it is difficult to bring about large changes to the clustering. For example, splitting a large cluster into two or merging two similar clusters into one might require stepping through intermediate clusterings with low probability.

One way to address the slow mixing issue is to use split-merge algorithms (Jain & Neal, 2000; Dahl, 2003a), which relies on Metropolis-Hastings proposals to merge two distinct clusters into one or split an existing cluster into two. In the following section, we present a different approach to speeding up sampling by augmenting the sampler with an auxiliary variable representing a permutation.

2.2. A combinatorial perspective

To develop our permutation-augmented sampler, we consider a combinatorial view of the DP mixture model. The prior over clusterings $\mathbf{C}$ induced by the Dirichlet process can be expressed compactly as follows (Antoniak, 1974):

$$p(\mathbf{C}) = \frac{\alpha_0^{|\mathbf{C}|}}{\mathcal{AF}(\alpha_0, n)} \prod_{c \in \mathbf{C}} (|c| - 1)!, \tag{1}$$

where $\mathcal{AF}(\alpha_0, n) = \alpha_0(\alpha_0 + 1) \cdots (\alpha_0 + n - 1)$ is the ascending factorial function.

Conditioned on the clustering $\mathbf{C}$, the marginal proba-

bility of the data $\mathbf{x}$ is given as follows:

$$p(\mathbf{x} \mid \mathbf{C}) = \prod_{c \in \mathbf{C}} \underbrace{\int \prod_{i \in c} F(x_i; \theta) G_0(d\theta)}_{\stackrel{\text{def}}{=} p(\mathbf{x}_c)}. \quad (2)$$

Note that given the clustering, the data points in each cluster are generated independently, a property that will be exploited in Section 3. By marginalizing out the mixture parameters θ, we obtain a function $p(\mathbf{x}_c)$ that depends only on the sufficient statistics of the points in cluster c.[1]

The goal of inference is to compute the posterior over clusterings:

$$p(\mathbf{C} \mid \mathbf{x}) = \frac{p(\mathbf{C}) p(\mathbf{x} \mid \mathbf{C})}{\sum_{\mathbf{C}'} p(\mathbf{C}') p(\mathbf{x} \mid \mathbf{C}')}.$$

This quantity involves a normalization constant which cannot be computed in practice because of the exponential number of possible clusterings. In the following section we show how our augmentation allows this difficulty to be circumvented.

3. Augmenting with a permutation

While we cannot hope to efficiently sum over all possible clusterings, we can efficiently sum over all clusterings in an appropriately constrained subset. If we order our data points and only consider *consistent* clusterings, i.e., clusterings that partition this ordering into contiguous segments, then it turns out to be possible to sum over clusterings efficiently using dynamic programming.

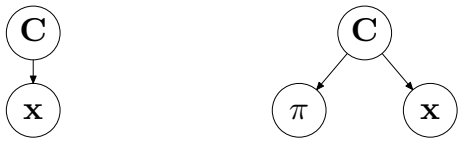
We thus augment the DP mixture model with a *permutation* $\pi = (\pi_1, \ldots, \pi_n)$, where x_{π_i} is the ith point in the permutation (Figure 1). For now, let $p(\pi \mid C)$ be the uniform distribution over consistent permutations:[2]

$$p(\pi \mid \mathbf{C}) = \frac{\mathbb{1}[\pi \in \Pi(\mathbf{C})]}{|\Pi(\mathbf{C})|} = \frac{\mathbb{1}[\pi \in \Pi(C)]}{|\mathbf{C}|! \prod_{c \in \mathbf{C}} |c|!}, \quad (3)$$

where $\Pi(\mathbf{C})$ is the set of permutations consistent with $\mathbf{C}$. In this augmented model, we can run a blocked Gibbs sampler where we alternate between sampling π given $(\mathbf{C}, \mathbf{x})$ and sampling $\mathbf{C}$ given $(\pi, \mathbf{x})$. Sampling π is easy: choose a random permutation of the clusters; for each cluster, choose a random permutation of the points in the cluster. Sampling $\mathbf{C}$ turns out to be also tractable via dynamic programming.

[1] This computation can be done easily in closed form if G_0 and F form a standard conjugate pair.

[2] In Section 4, we consider generating permutations from non-uniform distributions, in particular ones that depend on the data $\mathbf{x}$.



(a) DP mixture model (b) Permutation-augmented

Figure 1. We introduce an auxiliary variable for the permutation π and perform sampling in the augmented model.

3.1. Sampling a clustering using dynamic programming

We now present the dynamic program for sampling the clustering $\mathbf{C}$ given the permutation π and data $\mathbf{x}$. Combining Equations 1, 2, and 3 we obtain the following joint distribution:

$$p(\mathbf{C}, \pi, \mathbf{x}) = \underbrace{\frac{\alpha_0^{|\mathbf{C}|}}{\mathcal{AF}(\alpha_0, n)|\mathbf{C}|!}}_{\stackrel{\text{def}}{=} A(|\mathbf{C}|)} \prod_{c \in \mathbf{C}} \underbrace{\frac{p(\mathbf{x}_c)}{|c|}}_{\stackrel{\text{def}}{=} B(c)} \quad (4)$$

if $\pi \in \Pi(\mathbf{C})$ and 0 otherwise.

In order to sample from $p(\mathbf{C} \mid \pi, \mathbf{x})$, we need to compute the normalization constant:

$$p(\pi, \mathbf{x}) = \sum_{\mathbf{C}: \pi \in \Pi(\mathbf{C})} A(|\mathbf{C}|) \prod_{c \in \mathbf{C}} B(c).$$

Unlike $\prod_{c \in \mathbf{C}} B(c)$, $A(|\mathbf{C}|)$ does not decompose into a product of per-cluster factors. It requires global information—namely the number of clusters. Nonetheless, we can still apply dynamic programming *conditioned* on the number of clusters:

$$p(\pi, \mathbf{x}) = \sum_{K=1}^{n} A(K) \underbrace{\sum_{\mathbf{C}: \pi \in \Pi(\mathbf{C}), |\mathbf{C}| = K} \prod_{c \in \mathbf{C}} B(c)}_{\stackrel{\text{def}}{=} g(n, K)}.$$

The quantity $g(r, K)$ is a sum over all clusterings of the first r data points (with respect to the permutation π) with exactly K clusters. We compute this value recursively by summing over all possible sizes m of the last cluster:

$$g(r, K) = \sum_{m=1}^{r} g(r - m, K - 1) B(\{\pi_{r-m+1}, \ldots, \pi_r\}).$$

Given the size m, we sum recursively over clusterings of the first $r - m$ points and account for the points in the last cluster using the function B.

After computing all the entries in the dynamic programming table $g(r, K)$, we can sample a clustering by following the recurrence. Starting at $r = n$, we select a size m for the last cluster with probability proportional to its contribution $g(r-m, K-1)B(\{\pi_{r-m+1}, \ldots, \pi_r\})$ to the summation, then with $r = n - m$, and so on.

It is interesting to note that a clustering is sampled with probability proportional to $p(\mathbf{C}, \pi, \mathbf{x})$, which includes $p(\pi \mid \mathbf{C})$. This fact has an intuitive interpretation. In particular, the $p(\pi \mid \mathbf{C})$ factor down-weights clusterings with either very few or very many clusters—exactly those that are consistent with a large number of permutations. These clusterings would be oversampled without the $p(\pi \mid \mathbf{C})$ weighting.[3]

It is also worth noting that the augmentation method that we have described can be applied to a broad class of models beyond DP mixtures. Our approach applies as long as the prior decomposes into factors A and B as in Equation 4. In particular, the approach can be used for finite mixture models and mixture models based on Pitman-Yor process (Pitman & Yor, 1997).

The basic permutation-augmentation that we have introduced in this section requires $O(n^2)$ space and $O(n^3)$ time to sample a clustering. While the benefit obtained for this computation is a potentially large move, the computational burden is overly large to make this basic approach feasible in general. We thus turn to the discussion of several optimizations that we have developed that make the approach practical for large data sets.

3.2. Optimization 1: Metropolis-Hastings

The time and space complexities for computing $p(\pi, \mathbf{x})$ arise because of the need to sum over all possible cluster sizes. The culprit is the $\frac{1}{|\mathbf{C}|!}$ factor in $A(|\mathbf{C}|)$. If we replace $|\mathbf{C}|!$ with $\beta^{|\mathbf{C}|}$, then we would be able to move this factor, along with $\alpha_0^{|\mathbf{C}|}$, into $B(c)$. Doing so results in an approximate joint distribution:

$$q_\beta(\mathbf{C}, \pi, \mathbf{x}) = \underbrace{\frac{1}{\mathcal{AF}(\alpha_0, n)}}_{\stackrel{\text{def}}{=} A'} \prod_{c \in \mathbf{C}} \underbrace{\frac{p(\mathbf{x}_c)\alpha_0}{|c|\beta}}_{\stackrel{\text{def}}{=} B'(c)}. \qquad (5)$$

Note that A' does not depend on $\mathbf{C}$, and the dependence on $\mathbf{C}$ factors according to the clusters. This allows us to compute $q_\beta(\pi, \mathbf{x})$ using a much simpler dynamic program, one which requires only $O(n)$ space and $O(n^2)$ time:

$$g'(r) = \sum_{m=1}^{r} g'(r-m)B'(\{\pi_{r-m+1}, \ldots, \pi_r\}). \qquad (6)$$

We can sample from the approximate distribution $q_\beta(\mathbf{C} \mid \pi, \mathbf{x})$ rather than our desired distribution $p(\mathbf{C} \mid \pi, \mathbf{x})$. This introduces a bias, but we can correct for this bias using Metropolis-Hastings. Specifically, we accept a new clustering $\mathbf{C}_{\text{new}} \sim q_\beta$ with probability

$$\min\left\{1, \frac{p(\mathbf{C}_{\text{new}}, \pi, \mathbf{x})q_\beta(\mathbf{C}_{\text{old}}, \pi, \mathbf{x})}{p(\mathbf{C}_{\text{old}}, \pi, \mathbf{x})q_\beta(\mathbf{C}_{\text{new}}, \pi, \mathbf{x})}\right\}.$$

The only issue now is to choose β so that p and q_β are not very far apart. Our solution is to Taylor expand the log of the factorial function around the current number of clusters: $\log|\mathbf{C}_{\text{new}}|! \approxeq \log\Gamma(|\mathbf{C}_{\text{old}}| + 1) + \Psi(|\mathbf{C}_{\text{old}}| + 1)(|\mathbf{C}_{\text{new}}| - |\mathbf{C}_{\text{old}}|)$, where $\Psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$ is the digamma function, the derivative of $\log\Gamma(x)$. Letting $\beta = \exp\Psi(|\mathbf{C}_{\text{old}}|)$, we have $|\mathbf{C}_{\text{new}}|! \approxeq a\beta^{|\mathbf{C}_{\text{new}}|+1}$, where a is a constant which can be absorbed into A'. The approximation is good when we expect the distribution of $|\mathbf{C}|$ to be concentrated around the current number of clusters.

While this approach yields a significant reduction in complexity, it is important to note that the adaptation of β yields a sampler that does not necessarily have the correct stationary distribution. In practice, we adapt β only during the burn-in phase of the Markov chain. Thereafter, we fix β to the average value of $|\mathbf{C}|$ observed during burn-in.

3.3. Optimization 2: using a beam

The Metropolis-Hastings optimization yields a sampling algorithm that has a complexity of $O(n^2)$ time per sample. We now develop a second optimization that improves the running time to roughly $O(n)$. This optimization is motivated by two empirical observations. First, the sum for computing $g'(r)$ (Equation 6) is dominated by only a few terms (most of the time just a single term). Second, the vector of terms in the summation for computing $g'(r)$ (which we call the *summation profile*) and that of $g'(r+1)$ are usually very similar. Figure 2 shows the summation profile for several values of r.

The first observation suggests representing the summation profile of $g'(r)$ by a small subset of cluster sizes $M_r \subset \{1, \ldots, r\}$ sufficient to capture at least $1 - \epsilon$ of the full sum. Then we are guaranteed to lose at most a $1 - (1 - \epsilon)^n$ fraction of $g'(n)$.

[3] Note that in the setting of Bayesian network structural inference, a similar weighting is needed to guard against a bias towards structures with fewer edges. Friedman and Koller (2000) omit this weighting, resulting in a bias.

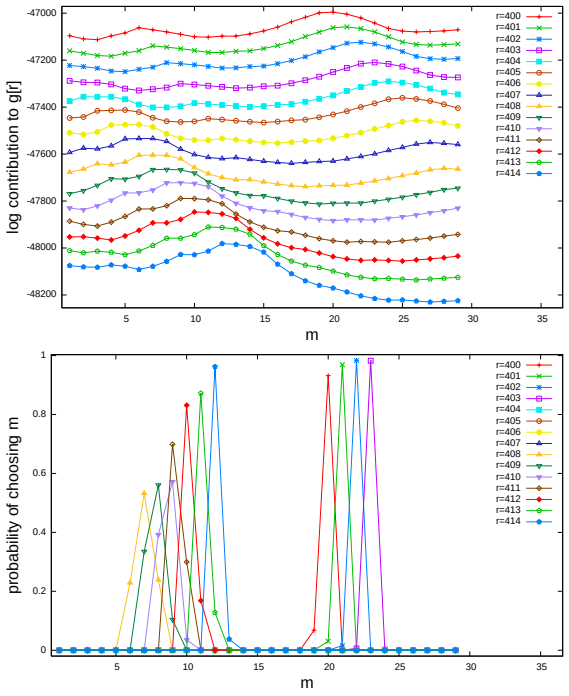


Figure 2. An example of the summation profile (the contribution $g'(r-m)B'(\{\pi_{r-m+1},\ldots,\pi_r\})$ as m varies) for several values of r. Top: log of the contribution. Bottom: the contribution normalized by $g'(r)$. For each r, we wish to find a few values of m that have large contributions. Since small differences in log space (a) are magnified in (b), finding the values of m with the largest contributions is crucial.

The second observation suggests computing the sets $M_1, M_2, \ldots$ incrementally as follows: given M_r, let M_{r+1} be the smallest subset of $N_r = \{m+1 : m \in M_r\} \cup \{1\}$ that captures at least $1-\epsilon$ of the sum over N_r. This procedure is heuristic because we choose M_r based on N_r rather than $\{1,\ldots,r\}$.

Empirically, we have found this approximation to be quite effective. On a data set with $n = 500$ points and $\epsilon = 10^{-32}$, the approximate $g'(n)$ was 0.97 of the true $g'(n)$. This was obtained by keeping $|M_r| \approxeq 5$ cluster sizes instead of the 250 that would have been required for the full $O(n^2)$ computation.

4. Data-dependent permutations

Thus far we have focused on the problem of sampling the clustering; we now turn to the problem of sampling the permutation. We propose doing this in a data-dependent way. Intuitively, "similar" points should be placed next to each other in the permutation, so that they will have a better chance of ending up in the same cluster when we sample the clustering. We present two ways of fleshing out this intuition: the first is based on constructing a permutation sequentially; the second is based on random projections.

4.1. Markov Gibbs scans

If we want to leverage dynamic programming when sampling the clustering, we must take care to generate a permutation π so that the factors of $p(\pi \mid \mathbf{C})$ respect the $A(|\mathbf{C}|)\prod_{c\in\mathbf{C}} B(c)$ template.

With that in mind, we generate the permutation as follows: we first choose one of the $|\mathbf{C}|!$ cluster permutations uniformly at random.[4] Then for each cluster $c \in \mathbf{C}$, we choose the first point uniformly from c. We choose the next point with probability proportional to the predictive likelihood given the previous point, and repeat until all points in the cluster have been chosen.

This particular choice is motivated by the fact that it encourages similar points to appear next to each other. It is simple to evaluate the likelihood $p(\pi \mid \mathbf{C})$, which is needed exactly (not just up to a normalization constant) for sampling $\mathbf{C}$ given π. The running time unfortunately increases from $O(|c|)$ to $O(|c|^2)$, which means that this technique will be effective only when clusters are moderately-sized.

4.2. Random projections

Random projections are a powerful technique that allow one to project n high-dimensional data points into $O(\log n)$ dimensions and preserve the pairwise distances (Johnson & Lindenstrauss, 1984). We use this to technique to project the data points down to one dimension, which induces a random permutation on the points. A vectorial representation of our data is needed, which is natural in most common cases (e.g., Gaussian or multinomial data). Specifically, we generate the permutation as follows:

- Choose a random unit vector u.
- For each cluster $c \in \mathbf{C}$, let v_c be the vectorial representation of the average sufficient statistics of $\mathbf{x}_c$ (e.g., the mean for Gaussian data).
- Sort the clusters by increasing $u \cdot v_c$.
- For each cluster $c \in \mathbf{C}$:
  - Let v_i be the vectorial representation of x_i.
  - Sort the data points $i \in c$ by increasing $u \cdot v_i$.

[4] A non-uniform distribution that tends to place similar clusters next to each other would be preferable, but then $p(\pi \mid \mathbf{C})$ would not be amenable to dynamic programming.

Unlike in the case of Markov Gibbs scans, it is a non-trivial task to compute $p(\pi \mid \mathbf{C})$, which is the proportion of the unit ball $\{u : ||u|| \leq 1\}$ that would produce the permutation π (characterized by linear constraints such as $u \cdot v_{\pi_1^c} < \cdots < u \cdot v_{\pi_{|c|}^c}$). This would require computing the volume of a convex body.[5]

Nonetheless, as we will show in Section 5, random projections can be very effective during the initial burn-in phase of the Markov chain for reaching good clusterings. For this phase, we propose the following stochastic hill-climbing algorithm: alternate between sampling from $p(\pi \mid \mathbf{C}, \mathbf{x})$ using random projections and sampling a clustering $\mathbf{C}$ with probability proportional to $p(\mathbf{C}, \mathbf{x})\mathbb{1}[\pi \in \Pi(\mathbf{C})]$. An advantage of this approach is that $p(\mathbf{C}, \mathbf{x})\mathbb{1}[\pi \in \Pi(\mathbf{C})]$ does not depend on $|\mathbf{C}|$, so we can forgo the first optimization (Section 3.2). After burn-in, we can switch to the data-independent or Markov Gibbs scan permutation sampling schemes to explore the state space in an unbiased manner.

5. Experiments

All ergodic samplers converge to the correct posterior over clusterings; the important question is how fast this happens. A typical sample path of a Markov chain can be broken into two phases, a *burn-in phase* and a *mixing phase*. A prerequisite to convergence is visiting high probability regions of the state space. During the burn-in phase, the sampler essentially does stochastic hill-climbing starting from a low probability clustering. In practice, when immediate progress has ceased (regardless of whether the chain has actually converged) the mixing phase begins, in which samples are used to compute averages of desired quantities. Our experiments show that using permutation augmentation can help quite a bit during the burn-in phase (Section 5.1) and is competitive in the mixing phase (Section 5.2).

We compared our permutation-augmented sampler with Gibbs (Escobar & West, 1995) and split-merge (Dahl, 2003a). Since neither of the samplers work well alone, we formed the following hybrids for comparison: Gibbs, Gibbs+SplitMerge, Gibbs+Perm, and Gibbs+SplitMerge+Perm. For the samplers that use more than one type of move, we interleaved the moves, dividing the CPU time roughly evenly among them.

After each iteration of sampling, we also sample the concentration parameter α_0 (West, 1995). We have devised a way of integrating out α_0 as part of the dynamic program for sampling clusterings; we omit the details for reasons of space.

[5] A general $O(n^5)$ algorithm exists for this task (Kannan et al., 1997) but is impractical for our purposes.

5.1. Burn-in phase

We use the stochastic hill-climbing permutation-augmented algorithm with random projections (Section 4.2) during the burn-in phase. Though biased, sampling the permutation in this data-dependent way is crucial for performance.

Synthetic data We generated 10,000 points from a finite mixture of Gaussians with a $\mathcal{N}(0, 2)$ prior on the mean (but used $\mathcal{N}(0, 100)$ during inference) and identity covariance matrix. To understand the regimes in which the various algorithms perform well, we varied three settings: sampler initialization, number of true clusters, and number of dimensions. Figure 3(a)–(h) demonstrates the rates of convergence of the different samplers across these conditions.

An often neglected issue is the question of initialization, which can impact performance significantly. For example, when initialized with a single cluster containing all the data points, Gibbs cannot easily form new clusters. On the other hand, initializing with too many clusters makes both Gibbs and split-merge inefficient: Gibbs takes $O(n^2)$ time per iteration and since split-merge proposes merging clusters at random, it will reject most of the time. As Figure 3(a)–(c) shows, Gibbs+Perm and Gibbs+SplitMerge+Perm, the two samplers with permutation moves, are largely insensitive to the type of initialization. We see that Gibbs+SplitMerge catches up slowly, but its performance degrades with more initial clusters (c). Gibbs remains stuck in a local optima, although it does much better with more initial clusters.

For all of the remaining experiments, we initialized using sequential prediction: a clustering is sampled according to the CRP but weighting the probability of choosing a cluster by the predictive likelihood of the new point given the existing points in the cluster.

Figure 3(d)–(f) shows the performance on 40-dimensional data for different numbers of true clusters. This experiment confirms our intuition that having more true clusters worsens the convergence of samplers with split-merge moves but does not affect the samplers with permutation moves.

Figure 3(g)–(h) shows the performance for 10 and 80 dimensional data sets with 40 true clusters. The gains resulting from permutation moves are sharpened in higher dimensions (h). We speculate that this is because random projections are more effective at separating out the true clusters in this regime.

Real-world data To test the samplers on real-world data, we took 10,000 handwritten digits from the MNIST data set (collected by Yann LeCun), normalized the data, and ran PCA to reduce the dimensionality from 28×28 to 50. The covariance of the DP mixture model is fixed to $16I$. Figure 3(i) shows that using permutation moves improves performance.

For multinomial data, we clustered 2246 AP news documents (collected by David Blei). Each document is represented as a 10,473-dimensional vector of word counts. Figure 3(j) shows that GIBBS+PERM underperforms GIBBS+SPLITMERGE, but is competitive when combined with split-merge (GIBBS+SPLITMERGE+PERM). We suspect that random projections are less effective at separating AP data, which consists of raw word counts and overlapping topics.

5.2. Mixing phase

We now evaluate the ability of the four samplers to mix after burn-in. We took 1000 examples from the MNIST data set and reduced the dimensionality to two.[6] Instead of using random projections, we sample the permutation uniformly in order to sample from the exact posterior.

Figure 3(k)–(l) shows the autocorrelation[7] of two statistics: the number of clusters and largest cluster size. Smaller values of autocorrelation indicate faster mixing. We see that for the number of clusters, there is little difference between the various algorithms. In the case of the largest cluster size, the samplers that include split-merge moves mix faster.

6. Discussion

We have presented a new sampling algorithm for DP mixtures by introducing an auxiliary variable representing the permutation of the data points. This enables us to use dynamic programming to sum over an exponential number of clusterings. We also proposed the use of random projections to yield an efficient approach to sampling the permutation.

Our method can be extended to hierarchical Dirichlet processes by defining a global permutation on the data points in all groups. In more complex models such as these, local optima are likely to be a bigger issue.

The permutation augmentation defines a tractable subset of clusterings which can be summed over efficiently. It would also be worthwhile to explore other types of augmentation schemes that yield tractable subproblems. For example, trees might define richer subsets of clusterings. As DP mixtures become more prevalent, the need for fast inference procedures becomes more pressing. This work is the first step in a new direction, which we hope will lead to many more algorithms.

[6] We reduced to two dimensions in this experiment so that we could ensure that a chain converges to the stationary distribution. This allows us to accurately estimate the mean and variance of the two statistics for computing the autocorrelation.

[7] Autocorrelation at lag τ of a sequence $x_1, \ldots, x_n$ is defined as $\frac{1}{n-\tau-1} \sum_{i=1}^{n-\tau} \frac{(x_i-\mu)(x_{i+\tau}-\mu)}{\sigma^2}$, where μ and σ^2 are the mean and variance of $x_1, \ldots, x_n$.

References

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics*, *2*, 1152–1174.

Blei, D., & Jordan, M. I. (2005). Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, *1*, 121–144.

Dahl, D. B. (2003a). *An improved merge-split sampler for conjugate Dirichlet process mixture models* (Technical Report). University of Wisconsin.

Dahl, D. B. (2003b). *Modal clustering in a univariate class of product partition models* (Technical Report). University of Wisconsin.

Daume, H. (2007). Fast search for Dirichlet process mixture models. *Artificial Intelligence and Statistics*.

Escobar, M. D., & West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, *90*, 577–588.

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

Friedman, N., & Koller, D. (2000). Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Uncertainty in Artificial Intelligence* (pp. 201–210).

Heller, K. A., & Ghahramani, Z. (2005). Bayesian hierarchical clustering. *International Conference on Machine Learning*.

Ishwaran, H., & James, L. F. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, *96*, 161–173.

Jain, S., & Neal, R. (2000). *A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model* (Technical Report). University of Toronto.

Johnson, W., & Lindenstrauss, J. (1984). Extensions of Lipschitz maps into a Hilbert space. *Contemporary Mathematics*, *26*, 189–206.

Kannan, R., Lovasz, L., & Simonovits, M. (1997). Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *11*, 1–50.

Kurihara, K., Welling, M., & Teh, Y. W. (2007). Collapsed variational Dirichlet process mixture models. *International Joint Conference on Artificial Intelligence*.

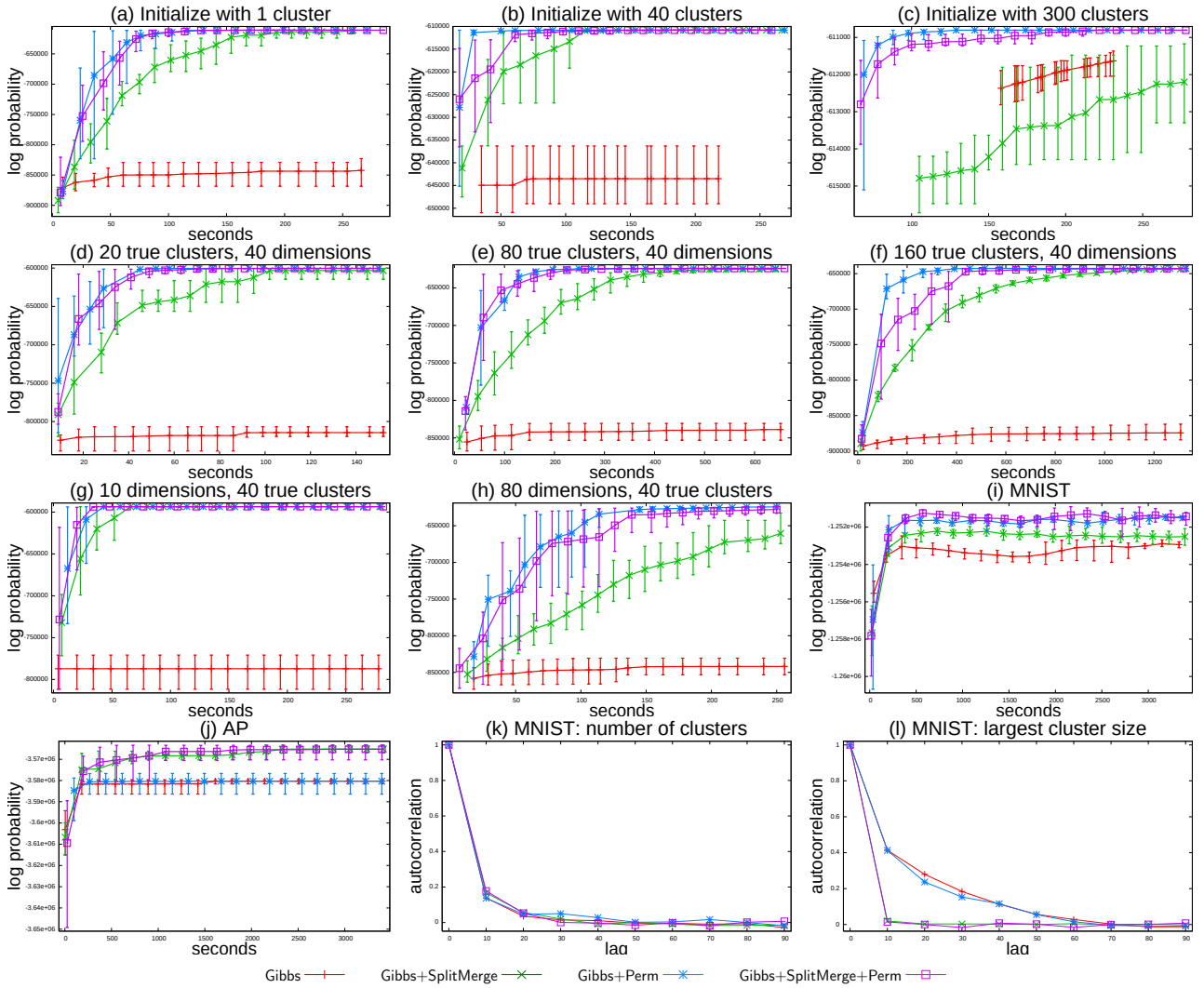


Figure 3. Plots (a)–(j) show the log probability during the burn-in phase for the four samplers on various data sets and initializations. Plots (k)–(l) show autocorrelations during the mixing phase. The error bars show the range of values obtained across five runs with different random seeds for both initialization and sampling.

Liu, J., & Wu, Y. (1999). Parameter expansion for data augmentation. *Journal of the American Statistical Association*, *94*, 1264–1274.

Pitman, J. (2002). *Combinatorial stochastic processes* (Technical Report 621). Department of Statistics, UC Berkeley.

Pitman, J., & Yor, M. (1997). The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, *25*, 855–900.

Sudderth, E. B., Torralba, A. B., Freeman, W. T., & Willsky, A. S. (2006). Describing visual scenes using transformed Dirichlet processes. *Advances in Neural Information Processing Systems* (pp. 1297–1304).

Swendsen, R. H., & Wang, J. S. (1987). Nonuniversal critical dynamics in mc simulations. *Physics Review Letters*, *58*, 86–88.

Tanner, M. A., & Wong, W. H. (1987). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, *82*, 528–540.

Teh, Y. W., Jordan, M., Beal, M., & Blei, D. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, *101*, 1566–1581.

West, M. (1995). *Hyperparameter estimation in Dirichlet process mixture models* (Technical Report). Duke University.

Xing, E. P., Sharan, R., & Jordan, M. I. (2004). Bayesian haplotype inference via the Dirichlet process. *International Conference on Machine Learning* (p. 111).