The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Scaling-Up Split-Merge MCMC with Locality Sensitive Sampling (LSS)

Chen Luo, Anshumali Shrivastava
Department of Computer Science, Rice University
{cl67, anshumali}@rice.edu

Abstract

Split-Merge MCMC (Monte Carlo Markov Chain) is one of the essential and popular variants of MCMC for problems when an MCMC state consists of an unknown number of components. It is well known that state-of-the-art methods for split-merge MCMC do not scale well. Strategies for rapid mixing requires smart and informative proposals to reduce the rejection rate. However, all known smart proposals involve expensive operations to suggest informative transitions. As a result, the cost of each iteration is prohibitive for massive scale datasets. It is further known that uninformative but computationally efficient proposals, such as random split-merge, leads to extremely slow convergence. This tradeoff between mixing time and per update cost seems hard to get around.

We leverage some unique properties of weighted MinHash, which is a popular LSH, to design a novel class of split-merge proposals which are significantly more informative than random sampling but at the same time efficient to compute. Overall, we obtain a superior tradeoff between convergence and per update cost. As a direct consequence, our proposals are around 6X faster than the state-of-the-art sampling methods on two large real datasets KDDCUP and PubMed with several millions of entities and thousands of clusters.

Introduction

Bayesian mixture models are of great interest due to their flexibility in fitting a countably infinite number of components which can grow with the data (Medvedovic, Yeung, and Bumgarner 2004). The growth of model complexity with the data is also in agreement with modern progress in machine learning over massive datasets. However, the appealing properties of Bayesian modeling come with hard computational challenges. Even with simple mixture models, the mathematical problems associated with training and inference are intractable. As a result, recent research focuses on developing tractable computational techniques. In particular, the use of Markov chain Monte Carlo (MCMC) methods, to sample from the posterior distribution (Andrieu et al. 2003; Nasrabadi 2007; Wang and Blei 2012) is widely prevalent. The practical utility of these methods is illustrated in several applications including haplotype reconstruction (Eronen, Geerts, and Toivonen 2003), nucleotide substitutions (Huelsenbeck and Ronquist 2001),and gene expression (Sharma and Adlakha 2015), etc.

Metropolis-Hastings (MH) (Andrieu et al. 2003) is a favorite class of MCMC methods, which includes several state-of-the-art algorithms that have proven useful in practice. MH is associated with a transition kernel which provides a proposal step. This step is followed by appropriate stochastic acceptance process that ensures detailed balance. A notable example of MH is the Split-Merge MCMC algorithm (Jain and Neal 2004; Wang and Russell 2015) which is particularly useful for problems where an MCMC state can be thought of as consisting of a number of components (or clusters). Here as the name suggests, the proposal step comprises of either a split or a merge. A split move partitions an existing mixture component (or cluster) into two, while a merge move combines two mixture components into one.

In the seminal work of (Jain and Neal 2004), split-merge MCMC procedure was proposed. To illustrate the process, the authors first introduce a random split-merge MCMC, where the split and the merge decision were taken uniformly at random. However, it was also pointed out, in the same paper, that due to the random nature of the proposal it was unlikely to lead to a new state x' with higher likelihood $\mathcal{L}(x')$ leading to low acceptance. To mitigate the slow progress, the authors then propose the restricted Gibbs split-merge (RGSM). In RGSM, the idea was to use restricted Gibbs sampling to generate proposals with a higher likelihood of acceptance, instead of a random proposal. Thus, a less number of MCMC iterations were sufficient for convergence due to fewer rejections. However, the cost of restricted Gibbs is very high. As a result, even though the iterations are less, each iteration is costly making the overall algorithm slow, especially for large datasets. Our experiments confirm this slow convergence of RGSM.

An essential and surprising observation about space asymmetry with smart proposals in split-merge MCMC was made in (Wang and Russell 2015). The authors show the necessity to mix smart and dumb (random) proposals for faster progress. They proposed a Smart-Dumb/Dumb-Smart Algorithm (SDDS) as an alternative to RGSM. Instead of relying on Gibbs sampling, the SDDS algorithm uses the likelihood of the model itself as a guiding strategy for smart proposals. In other words, the SDDS method evaluates a large number of possible proposals x' based on the likelihood of each

x' and choose the best ones. This strategy, as expected, ensures a higher chance of improving the state x with every proposal. However, from a computational perspective, it is not difficult to see that smart proposal x' obtained after evaluation of a large number of proposal states, based on the likelihood, is equivalent to evaluating all these states for acceptance/rejection as part of MH (Wang and Russell 2015). As a result, the reduction in the number of iteration is not helpful in obtaining an efficient algorithm. Our experiments show that SDDS also has poor convergence.

Unfortunately, most MCMC methodologies ignore the tradeoff between the number of iteration and computations associated with each iteration. They instead only focus on reducing the number of rejections, which is often achieved by informative proposals with increased per iteration cost. In this paper, we are interested in efficient split-merge MCMC algorithm which leads to overall fast convergence. Thus, reducing both is the aim of this work.

Parallelization is Complementary: Due to the significance of the problem there are several works which try to scale up MCMC by using parallelism. Parallelism is often achieved by running parallel MCMC chains on subsets of data and later merging them (Chang and Fisher III 2013). Since our proposal reduces the overall cost of split-merge MCMC algorithm in general, it will reduce the cost of each of the parallel chains thereby increasing the effectiveness of these parallelisms on MCMC. Thus, existing advances in parallelizing MCMC is complementary to our proposal.

Our Contributions: In this work, we show that it is possible to construct informative proposals without sacrificing the per-iteration cost. We leverage a simple observation that while designing proposals we can favor configurations where entities similar are likely to be in the same component. We use standard notions of vector similarity such as cosine or Weighted jaccard. To perform such sampling efficiently, we capitalize on the recent advances in LSH sampler (Luo and Shrivastava 2018; Spring and Shrivastava 2017a; Charikar and Siminelakis 2017) that can perform adaptive sampling based on similarity. This forms our first proposal.

Our first proposal leads to around 3x improvements over state-of-the-art methods. However, with similarity driven sampling, computing the Metropolis-Hastings (MH) ratio requires quadratic cost in the size of the cluster being split or merged. This is because while computing the state transition probability, we need to evaluate all possible ways that can lead to the desired split configuration. All these configurations have different probabilities due to similarity-based adaptive sampling and hence the probability computation is expensive. It appears at first that this cost is unavoidable. Surprisingly, it turns out that there is a rare sweet spot. With Weighted MinHash, we can design a split-merge proposal where the total cost of MH update is only linear in the size of the cluster being split or merged. The possibility is unique to MinHash due to its k-way generalized collision probability (Shrivastava and Li 2013). Our proposal and novel extension of MinHash collision probability could be of independent interest in itself.

Overall, our proposed algorithms obtain a sweet tradeoff between the number of iteration and computational cost per iteration. As a result, we reduce the overall convergence in time, not just in iterations. On two large public datasets, our proposal MinHash Split-Merge (MinSM) significantly outperforms other state-of-the-art split-merge MCMC algorithms in convergence speed as measured on wall clock time on the same machine. Our proposed algorithm is around 6x faster than the second best baseline on synthetic datasets as well as realworld datasets without loss in accuracy.

Background

Our work requires bridging Locality Sensitive Sampling with split-merge MCMC algorithm. We briefly review the necessary background.

Locality Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a popular technique for efficient approximate nearest-neighbor search. LSH is a family of functions, such that a function uniformly sampled from this hash family has the property that, under the hash mapping, similar points have a high probability of having the same hash value. More precisely, consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to a discrete set $[0, R-1]$.

Definition 1 Locality Sensitive Hashing (LSH) Family *A family $\mathcal{H}$ is called (S_0, cS_0, u_1, u_2)-sensitive if for any two points $x, y \in \mathbb{R}^d$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if $Sim(x, y) \geq S_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \geq u_1$*
- *if $Sim(x, y) \leq cS_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \leq u_2$*

A collision occurs when the hash values for two data vectors are equal, meaning that $h(x) = h(y)$. LSH is a very well studied topic in computer science theory and database literature. There are many well-known LSH families in the literature. Please refer (Gionis et al. 1999) for details.

Locality Sensitive Sampling (LSS) and Unbiased Estimators LSH was considered as a black-box algorithm for similarity search and dimensionality reduction. Recent research (Spring and Shrivastava 2017a; Charikar and Siminelakis 2017; Luo and Shrivastava 2018; Chen, Shrivastava, and Steorts 2017; Spring and Shrivastava 2017b) found that LSH can be used for something more subtler but useful. It is a data structure that can be used for efficient dynamically adaptive sampling. We first describe the sampling algorithm of (Spring and Shrivastava 2017a; Charikar and Siminelakis 2017; Luo and Shrivastava 2018; Chen, Shrivastava, and Steorts 2017) and later comment on its properties crucial to our proposal.

The algorithm uses two parameters - (K, L). We construct L independent hash tables from the collection $\mathcal{C}$. Each hash table has a meta-hash function H that is formed by concatenating K random independent hash functions from some appropriate locality sensitive hash family $\mathcal{H}$. The candidate sampling algorithm works in two phases (Spring and Shrivastava 2017a; Charikar and Siminelakis 2017; Luo and Shrivastava 2018; Chen, Shrivastava, and Steorts 2017): (1) **Pre-processing Phase:** We construct L hash tables from the data by storing all elements $x \in \mathcal{C}$. This is

one-time linear cost. (2) **Sampling Phase:** Given a query q, we collect one bucket from a randomly selected hash table and return a random element from the bucket. If the bucket is empty, we reselect a different hash table again. Keep track of the number of different tables T probed.

It is not difficult to show that an item returned as a candidate from a (K, L)-parameterized LSH algorithm is sampled with probability exactly $1-(1-p^K)^L \times \frac{1}{Size}$, where p is the collision probability of LSH function and $Size$ is the number of elements in the bucket (Spring and Shrivastava 2017a; Charikar and Siminelakis 2017; Luo and Shrivastava 2018; Chen, Shrivastava, and Steorts 2017). The LSH family defines the precise form of p used to build the hash tables. Specifically, when $L = 1$ and $K = 1$, the probability reduced to the collision probability itself (p). Our proposal will heavily rely above observation to design an informative proposal distribution.

Weighted (or Generalized) MinHash

Weighted Minwise Hashing is a known LSH for the Weighted Jaccard similarity (Leskovec, Rajaraman, and Ullman 2014). Given two positive vectors $x,\ y \in \mathbb{R}^{\mathbb{D}}, x,\ y > 0$, the (generalized) Weighted Jaccard similarity is defined as $\mathbb{J}(x, y) = \frac{\sum_{i=1}^{D} \min\{x_i, y_i\}}{\sum_{i=1}^{D} \max\{x_i, y_i\}}$., where $\mathbb{J}(x, y)$ is a frequently used measure for comparing web-documents (Leskovec, Rajaraman, and Ullman 2014), histograms, gene sequences, etc.

Weighted Minwise Hashing (WMH) (or Minwise Sampling) generates randomized hash (or fingerprint) $h(x)$, of the given data vector $x \geq 0$, such that for any pair of vectors x and y, the probability of hash collision (or agreement of hash values) is given by $Pr(h(x) = h(y)) = \frac{\sum \min\{x_i, y_i\}}{\sum \max\{x_i, y_i\}}$.

A unique property of Minwise Hashing is that there is a natural extension of k-way collision (Shrivastava and Li 2013). In particular, given vectors $x^{(1)},\ x^{(2)}, ..., x^{(s)}$, the simultaneous collision probability is given by:

$$Pr(h(x^{(1)}) = h(x^{(2)}) = ... = h(x^{(s)})) = \frac{\sum_j^D \min\{x_j^{(1)}, x_j^{(2)}, ..., x_j^{(s)}\}}{\sum_j^D \max\{x_j^{(1)}, x_j^{(2)}, ..., x_j^{(s)}\}} \quad (1)$$

Minwise hashing can be extended to negative elements using simple feature transforms (Li 2017), which essentialy doubles the dimentions to 2D. In this paper, MinHash and Weighted MinHash denote the same thing.

Split-Merge MCMC

Split-Merge MCMC (Hughes, Fox, and Sudderth 2012) is useful for dealing with the tasks such as clustering or topic modeling where the number of clusters or components are not known in advance. Split-Merge MCMC is a Metropolis-Hastings algorithm with two main transitions: Split and Merge. During a split, a cluster is partitioned into two components. On the contrary, a merge takes two components and makes them to one.

During the MCMC inference process, split and merge moves simultaneously change the number of components and change the assignments of entities to different components. (Jain and Neal 2004) proposes the first non-trivial Restricted Gibbs Split-Merge (RGSM) algorithm, which was later utilized for efficient topic modeling over large datasets in (Wang and Blei 2012).

In (Wang and Russell 2015), the authors presented a surprising argument about information asymmetry. It was shown that both informative split and merge leads to poor acceptance ratio. The author proposed a combination of the smart split with dumb (random) merge and dumb split with smart merge as a remedy. The algorithm was named as Smart-Dumb/Dumb-Smart Split Merge algorithm (SDDS), which was superior to RGSM. To obtain non-trivial smart split (or merge), the authors propose to evaluate a large number of dumb proposals based on the likelihood and select the best. This search process made the proposal very expensive. It is not difficult to see that finding a smart split is computationally not very different from running a chain with several sequences of dumb (random) splits (Wang and Russell 2015).

LSS based Split-Merge MCMC

Utilizing Similarity Information: In this paper, we make an argument that similarity information, such as cosine similarity, between different entities is almost always available. For example, in the clustering task, the vector representation of the data is usually easy to get for computing the likelihood. Even in an application where we deal with complex entities such as trees, it is not uncommon to have approximate embeddings (Bengio, Weston, and Grangier 2010).

It is natural to believe that similar entities, in terms of cosine similarity or Jaccard distance, of the underlying vector representation, are more likely to go to the same cluster than non-similar ones. Thus, designing proposals which favor similar entities in the same cluster and dissimilar entities in different clusters is more likely to lead to acceptation than random proposals.

However, the problem is far from being solved. Any similarity based sampling requires computing all pairwise similarity as a prerequisite, which is a quadratic operation $O(n^2)$. Quadratic operations are near-infeasible for large datasets. One critical observation is that with the modern view of LSH as samplers, described preview section, we can get around this quadratic cost and design cheaper non-trivial proposals.

Naive LSS based Proposal Design

This section discusses how LSH can be used for efficient similarity sampling which will lead to an informative proposal. In addition, we also want the cost of computing the transition probabilities $q(x'|x)$, which is an important component of the acceptance ratio $\alpha(x'|x)$ (Jain and Neal 2004), to be small. Here, x denotes the state before split/merge, and x' denote the state after split/merge. For a good proposal design, it is imperative that $q(x'|x)$ is easy to calculate as well as the proposed state x' is informative. Thus, designing the right MCMC proposal process is the key to speed up computation. Following the intuition described before, we

introduce our LSS based proposal design in the rest of this section.

We first create the hash tables T for sampling. We use Sign Random Projection as the LSH function, thus our notion of similarity is cosine (Gionis et al. 1999). It is pointed out that, we can also use other LSH functions when the similarity notion is different. We pay a one-time linear cost for this preprocessing. Note, we need significantly less K and L (both has value 10 in our experiments) compared to what is required for near-neighbor queries as we are only sampling. The sampling is informative for any values of K and L. For the details of analysis on K and L, please refer (Spring and Shrivastava 2017a).

For our informative proposal, we will need capabilities to do both similarity sampling as well as dissimilarity sampling for merge and split respectively. The similarity sampling is the usual sampling algorithm discussed in previous sections, which ensures that given a query u, points similar to u are more likely to be sampled. Analogously, we also need to sample points that are likely to be dissimilar. With cosine similarity, flipping the sign of the query, i.e., changing u to $-u$ will automatically do dissimilarity sampling.

Inspired from (Wang and Russell 2015), we also leverage the information asymmetry and mix smart and dumb moves for better convergence. However, this time our proposals will be efficient. At each iteration of MCMC, we start by choosing randomly between an LSH Smart-split/Dumb-merge or an LSH Smart-merge/Dumb-split operation. These two operations are defined below:

Naive LSH Smart-split/Dumb-merge LSH based split begins by randomly selecting an element u in the dataset. Then, we use LSS (Locality-sensitive Sampler) to sample points likely to be dissimilar to u. Thus, we query our data structure T with $-u$ as the query to get another element v which is likely far away from u. If u and v belong to the same cluster C, we split the cluster. During the split, we create two new clusters C_u and C_v. We assign u to C_u and v to C_v. For every element in C, we randomly assign them to either C_u or C_v. Since we ensure that dissimilar points u and v are split, this is an informative or smart split. If we find u and v are already in a different cluster, we do a dumb merge: randomly select two components, and merge these two components into one component.

The most important part is that we can precisely compute the probability of the proposed split move $q(x'|x)$ and the corresponding inverse move probability $q(x|x')$ as follow:

$$
\begin{aligned}
q(x'|x) &= \left(\frac{1}{2}\right)^{|C_u|+|C_v|-2} \\
&\sum_{u}^{C_u}\sum_{v}^{C_v}\left(\frac{1}{n}\left(1-\left(1-Pr(-u,v)^K\right)^L\right)\frac{|C_v\cap S_{-u}|}{|S_{-u}|}\right). \\
&= \frac{\sum_{u}^{C_u}\sum_{v}^{C_v}\left(\frac{1}{n}\left(1-\left(1-Pr(-u,v)^K\right)^L\right)\frac{|C_v\cap S_{-u}|}{|S_{-u}|}\right)}{2^{|C_u|+|C_v|-2}} \\
q(x|x') &= \frac{2}{M_{x'}(M_{x'}-1)}.
\end{aligned}
\tag{2}
$$

In the above, n is the number of data point. S_{-u} is the set of data points that returned by querying in T using $-u$, and $|S_{-u}|$ denotes the number of elements in S_{-u}. $M_{x'}$ denotes the number of clusters in state x'. C denotes the original component, C_u and C_v are the two new components after split with elements u and v in them. K is the number of bits used for hashing, and L is the number of hash tables probed. $Pr(-u,v)$ is the collision probability between $-u$ and v.

Naive LSH Smart-merge/Dumb-split LSH based Merge begins by randomly selecting an element u in the dataset. Then use LSS to sample from hash tables T to get another element v which is similar with u. Then, if the mixture component of u and v are different, then we do merge operation for the corresponding two mixture component. If u and v are in the same components, we do a dumb split: randomly select one cluster, and split this component into two separate components.

We provide the the probability of the merge move $q(x'|x)$ and the corresponding inverse probability $q(x|x')$:

$$
\begin{aligned}
q(x'|x) &= \sum_{u}^{C_u}\sum_{v}^{C_v}\left(\frac{1}{n}\left(1-\left(1-Pr(u,v)^K\right)^L\right)\frac{|C_v\cap S_u|}{|S_u|}\right), \\
q(x|x') &= \frac{1}{M_{x'}}(\frac{1}{2})^{|C_u|+|C_v|}.
\end{aligned}
\tag{3}
$$

In the above, S_u is the set of data points that returned by query in T using u. $|S_u|$ denotes the number of elements in S^s. All the other symbols have the same meaning as before. $Pr(u,v)$ is the collision probability between u and v.

Notice that, to calculate the transition probabilities in Eq. 2 and Eq. 3, we need to sum over all possible u and v in the two components C_u and C_v. This could be expensive when the cluster size is large. In other words, this complexity of this proposal is quadratic to the size of the cluster.

The quadratic cost seems unavoidable. LSH does similarity based sampling. Thus, we can sample pairs u and v in adaptive fashion efficiently. A split of cluster C into C_u and C_v can happen because of any two elements $x \in C_u$ and $y \in C_v$ being samples. As a result, the transition probability requires accumulating non-uniform probabilities of all possible combinations, making it quadratic to compute. On the other hand if every pair has same probability then the proposal is random. Overall, it seems hopeless to split the cluster adaptively and at the same time get the probability of split linear in the size of cluster.

It turns out, surprisingly, that a very unique design of proposal that satisfies our wishlist. It is the unique mathematical properties of MinHash and a novel generalization of its k-way collision probability that makes this possible. In the next section, we will introduce the method of use k-way minhash for scaling up MCMC (Shrivastava and Li 2013).

MinSM: MinHash based Split-Merge MCMC

Ideally, after identifying u we should split so that all the elements similar to u goes to C_u and rest goes to C_v. This will be a significantly more informative proposal than random assignments to C_u and C_v. However, evaluating the transition

probability of configuration under LSH would be computationally expensive, as LSH sampling is correlated and the expressions are contrived as we introduced before.

We next show that MinHash with a very specific design exactly achieves this otherwise impossible and ideal state with the cost of evaluating the transition probability linear in the size of the cluster. A unique property of MinHash is that we can compute, in closed form and linear cost, the probability of collision of a set of points of any size ≥ 2. Such computation is not possible with any other know LSH including the popular random projections (Gionis et al. 1999).

We provide a novel extension of the collision probability of MinHash to also include the probability of collision with a given set and no collision with another given set (See Equation 1). It is surprising that despite many non-trivial correlations, the final probability expression is very simple and only requires linear computations. As a result, we can directly get the split of a cluster into two sets (or clusters) and at the same time compute the transition probability. The novel design and analysis of Minhash, presented here, could be of independent interest in itself.

MinHash Smart-split/Dumb-merge MinHash based split begins by flipping a coin to randomly choose from the action of Smart-split or Dumb merge.

The LSH smart-split begins by randomly selecting an element u in the dataset. Then, we use LSS (Locality-sensitive Sampler) to sample a set of points that are likely to be similar to u from T, i.e., query T with u. Here we use Weighted MinHash as the LSH and $K = 1$ is necessary. $K \geq 1$ makes the probability computations out of reach. Instead of sampling a point from the bucket, as we do with LSS, we just report the whole bucket as the set. Let us denote this sampled set as S_u. We now split the component C_u into two components: $C_u \cap S_u,\ C_u - S_u$. If the action is a dumb merge, then we randomly select two components and merge these two components into one component.

Given a new state x', and the corresponding old state x, we can precisely compute the probability of the proposed split move $q(x'|x)$ and the corresponding inverse move probability $q(x|x')$ as follow:

Define p as the probability of agreement of weighted minhash of u with all of the data point in the queried set S_u. The known theory (Gionis et al. 1999) says that the expression of p is given by Equation 1. However, we want something more, we want all elements of S_u to collide with u in the bucket and anything in $C_u - S_u$ to not collide. Define $Prob$ as the probability of agreement of weighted minhash of u with all of S_u and none of the data point in $C_u - S_u$. It turns our that we can calculate this probability exactly as:

$$Prob = \frac{\sum_j^{2D} \max\{0, (x^j_{\min} - x^j_{\max})\}}{\sum_j^{2D} x^j_{all}}, \tag{4}$$

where $x^j_{\min} = \min_{x \in C_u \cap S_u}\{x_j\}$, $x^j_{\max} = \max_{x \in C - S_u}\{x_j\}$ and $x^j_{all} = \max_{x \in C_u}\{x_j\}$

When we only use $K = 1$ Minhash, then the corresponding proposal distribution is shown as follow:

$$q(x'|x) = \frac{|S_u|}{n} \times Prob. \tag{5}$$

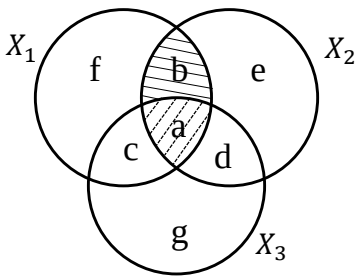


Figure 1: Three way Minwise Hashing.

Here, we give an illustration of the proof. Consider Figure 1. Let's start with vanilla MinHash over sets and the arguments will naturally extend to weighted versions. Given X_1, X_2 and X_3. We want the probability that the MinHash of X_1 and X_2 collide but not of X_3. From the theory of consistent sampling (Shrivastava and Li 2013; Shrivastava 2016; Manasse, McSherry, and Talwar 2010). This will happen if we sample from b and the possibility is the union. Thus the probability is $\frac{b}{a+b+c+d+e+f+g} = \frac{|X_1 \cap X_2| - |X_3|}{|X_1 \cup X_2 \cup X_3|}$ which is essentially we want the minimum of $|X_1 \cup X_2 \cup X_3|$ to be sampled from the intersection of X_1 and X_2 and not from X_3. That is the only way the MinHash of X_1 and X_2 will agree but not of X_3. This argument can be naturally extendend if we want $X_1, X_2, ..., X_h$ to have same minhash and not $Y_1, Y_2, ..., Y_g$, the probability can be written as:

$$\frac{\max\{0, |X_1 \cap X_2 \cap ... \cap X_h| - |Y_1 \cup Y_2 \cup ... \cup Y_g|\}}{|X_1 \cup X_2 \cup ... \cup X_h \cup Y_1 \cup Y_2 \cup ... \cup Y_g|}.$$

Now for weighted sets (non-binary), we can replace intersection with minimum and unions with max leading to the desired expression, which is due to the seminal works in consistent weighted sampling a strict generalization of MinHash. See (Shrivastava and Li 2013; Shrivastava 2016; Manasse, McSherry, and Talwar 2010) for details. Also using (Leskovec, Rajaraman, and Ullman 2014) we can extend it to negative weights as well using simple feature transformation.

It should be noted that this expression only requires cost linear in the size of the cluster C_u being split. With this value of $Prob$, the corresponding transition probability for the split move is:

$$q(x'|x) = \frac{|S_u|}{n} \times Prob, \quad q(x|x') = \frac{2}{M_{x'}(M_{x'} - 1)}. \tag{6}$$

In the above, n is the number of data point. S_u is the set of data points that returned by querying in T using u, and $|S_u|$ denotes the number of elements in S_u. D is the dimension of the data. $M_{x'}$ denotes the number of clusters in state x'.

To be able to compute this expression and also get an informative split was the primary reason for many choices that we made. For example, $K = 1$ as needed so that we can compute $Prob$ in a simple closed form. As a result, we obtain a very unique proposal. The idea and design could be of independent interest in itself.

Minhash Smart-merge/Dumb-split The proposed smart-merge begins by randomly selecting a center u in the dataset. Then, we use LSS (Locality-sensitive Sampler) to sample a center v that are likely to be similar to u. Then we merge the component C_u and C_v to one component.

If the action is a dumb split: randomly select one cluster, and split this component into two separate components uniformly.

Given a new state x', and the corresponding old state x. We provide the probability of the merge move $q(x'|x)$ and the corresponding inverse probability $q(x|x')$ as follow:

$$q(x'|x) = \frac{1}{M_x} \frac{\sum_j^{2D} \min\{u_j, v_j\}}{\sum_j^{2D} \max\{u_j, v_j\}} \frac{1}{|S_u|}, \\ q(x|x') = \frac{1}{M_{x'}} (\frac{1}{2})^{|C_u|+|C_v|}. \tag{7}$$

In the above, S_u is the set of data points that returned by the query in hash table T using u. $|S_u|$ denotes the number of elements in S_u. u_j denotes j-th feature of the data point u. All the other symbols have the same meaning as before.

As we introduced before, our proposed algorithm belongs to the general framework of metroplis-hastings algorithm (Andrieu et al. 2003). After each split/merge move, we need to calculate the acceptance rate $\alpha(x'|x)$ for this move: $\alpha(x'|x) = \min\{1, \frac{L(x')q(x|x')}{L(x)q(x'|x)}\}$, where x' is the proposed new state, x is the previous state, $q(x'|x)$ here is the designed proposal distribution, and it can be calculated as introduced in previous sections. $\mathcal{L}(x)$ is the likelihood value of the state x.

The likelihood of the data is generally in the form of $L(x) = \prod_D p_j(e_i)$, where $p_j(e_i)$ is the probability of $e_i \in D$ in it's corresponding component C_j. D denotes the total dataset. In the split merge MCMC, only the components that being split/merged will change of the likelihood value. So, that the ratio $\frac{L(x')}{L(x)}$ is cheap to compute, since all the probability of unchanged data will be canceled.

Empirical Study

In this section, we demonstrate the advantage of our proposed models by applying it to the Gaussian Mixture model inference and compare it with state-of-the-art sampling methods.

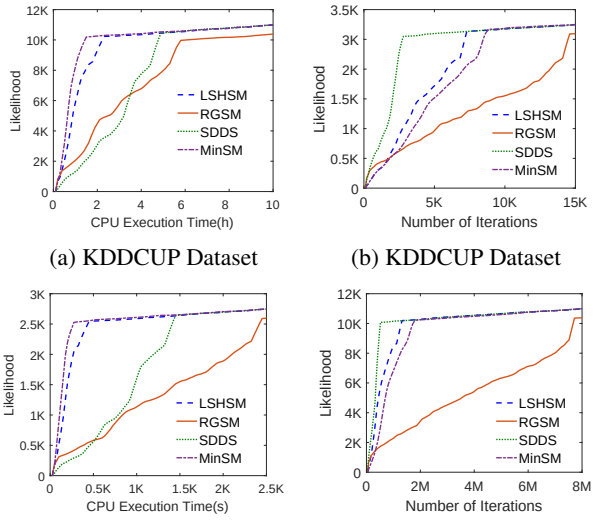
Gaussian Mixture Model

We briefly review the Gaussian Mixture Model. A Gaussian mixture density is a weighted sum of component densities. For a M-class clustering task, we could have a set of GMMs associated with each cluster. For a D-dimensional feature vector denoted as x, the mixture density is defined as $p(x) = \sum_{i=1}^{M} w_i p_i(x)$, where $w_i, i = 1, ..., M$ are the mixture weights which satisfy the constraint that $\sum_i^M = 1$ and $w_i \geq 0$. The mixture density is a weighted linear combination of component M uni-model Gaussian density functions $p_i(x), i = 1, ..., M$. The Gaussian mixture density is parameterized by the mixture weights, mean vectors, and covariance vectors from all components densities.

For a GMM-based clustering task, the goal of the model training is to estimate the parameters of the GMM so that the Gaussian mixture density can best match the distribution of the training feature vectors. Estimating the parameters of the GMM using the expectation-maximization (EM) algorithm (Nasrabadi 2007) is popular. However, in most of the real world applications, the number of clusters M is not known, which is required by the EM algorithm. On the other hand, Split-Merge based MCMC algorithms are used for inference when M is unknown, which is also the focus of this paper. We therefore only compare our proposal LSHSM and other state-of-the-art split-merge algorithms on GMM clustering which does not require the prior knowledge of the number of clusters.

Experimental Setup

Competing Algorithms: We compare following four split-merge MCMC sampling algorithm on GMM with an unknown number of clusters: **RGSM:** Restricted Gibbs split-merge MCMC algorithm (Jain and Neal 2004) is considered as one of the state-of-the-art sampling algorithm. **SDDS:** Smart-Dumb/Dumb-Smart Split Merge algorithm (Wang and Russell 2015). **LSHSM:** The Naive version of LSH based Split Merge algorithm by using Sign Random Projection. In the LSHSM method, we use fixed $K = 10$ and $L = 10$ for all the dataset. We fix the hashing scheme to be signed random projection. **MinSM:** LSH based split merge algorithm is the proposed method in this paper. In the MinSM method, we use fixed $K = 1$ and $L = 1$ for all the dataset.

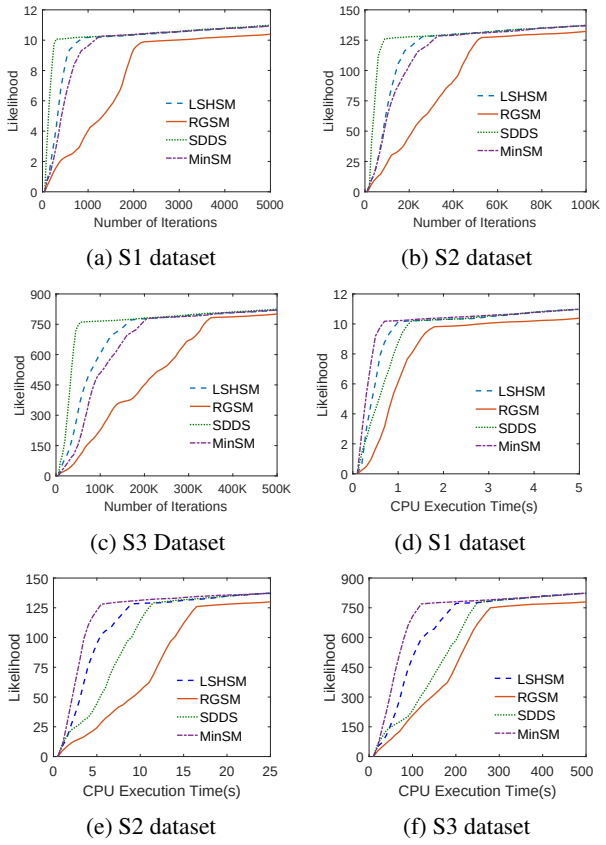


(a) KDDCUP Dataset (b) KDDCUP Dataset

(c) PubMed Dataset (d) PubMed Dataset

Figure 2: The time and iteration wise comparison of the likelihood for difference methods on the two real dataset. It is obviously that our proposed MinSM algorithm can be at least 6 times faster than the state of the art algorithms in the real large dataset.

Dataset: We evaluate the effectiveness of our algorithm on both two large real-world datasets: **KDDCUP** and



(a) S1 dataset (b) S2 dataset

(c) S3 Dataset (d) S1 dataset

(e) S2 dataset (f) S3 dataset

Figure 3: The time and iteration wise comparison of the likelihood for difference methods on the Synthetic Dataset. MinSM outperforms the other baselines by a large margin. It is also clear that requiring less iteration does not mean faster convergence.

PubMed. **KDDCUP** data was used in the KDD Cup 2004 data mining competition. It contains 145751 data point. The dimensionality of the dataset is 74. We have 2000 ground truth cluster labels for this dataset. [1] The **PubMed** abstraction dataset contains 8200000 abstractions that extracted from the PubMed [2]. All the documents represented as the bag-of-words representation. In the data set, we have 141043, different words. This data set is ideal for document clustering or topic modeling.

Synthetic data is a standard way of testing GMM models (Nasrabadi 2007). So, in this paper, we also use synthetic datasets as a sanity check to evaluate the performance of different methods. The process of generating the synthetic dataset is as follow: Randomly generate k different Gaussian distributions (with different corresponding mean and variance). We fix the $k = 10$ in our experiment. Then based on the randomly generated Gaussian distributions, we generate a set of data points for each Gaussian distribution. Here we fix the dimensionality of each data point to 25. In this experiment, we generate three sythntic dataset with different size (e.g. 100, 1000, 10000). We name the three synthetic dataset as **S1**, **S2**, **S3**.

[1] https://cs.joensuu.fi/sipu/datasets/

[2] www.pubmed.gov

Table 1: Clustering Accuracy for Different Methods

Methods	Metric	S1	S2	S3	KDD	Pub
RGSM	NMI	0.96	0.93	0.88	0.74	0.63
	Accuracy	0.95	0.92	0.87	0.68	0.62
SDDS	NMI	**0.97**	**0.96**	0.95	**0.86**	**0.80**
	Accuracy	**0.98**	**0.97**	0.94	**0.85**	**0.77**
LSHSM	NMI	0.96	0.95	**0.96**	0.84	0.77
	Accuracy	0.97	0.94	0.96	0.83	0.75
MinSM	NMI	0.96	0.94	**0.96**	0.83	0.75
	Accuracy	0.97	0.94	**0.97**	0.84	0.74

Speed Comparison and Analysis

We first plot the evolution of likelihood both as a function of iterations as well as the time of all the three competing methods. The evolution of likelihood and time with iterations on two real-world data is shown in Fig. 2. The result on three synthetic data set is shown in Fig. 3.

We can see a consistent trend in the evolution of likelihood, which holds true for both simulated as well as real datasets. First of all, RGSM consistently performs poorly and requires both more iterations as well as time. This demonstrate that the need of combining smart and dumb moves for faster convergence made in (Wang and Russell 2015) is necessary. RGSM does not use it and hence leads to poor, even iteration wise, convergence.

SDDS seems to do quite well, compared to our proposed LSHSM when we look at iteration wise convergence. However, when we look at the time, the picture is completely changed. MinSM is significantly faster than SDDS, even if the convergence is slower iteration wise. This is not surprising because the per-iteration cost of MinSM is orders of magnitude less than SDDS. SDDS hides the computations inside the iteration by evaluating every possible state in each iteration, based on likelihood, is equivalent to several random iterations combined. Such costly evaluation per iteration can give a false impressing of less iteration.

It is clear from the plots that merely comparing iterations and acceptance ratio can give a false impression of superiority. Time wise comparison is a legitimate comparison of overall computational efficiency. Clearly, MinSM outperforms the other baselines by a large margin.

Clustering Accuracy Comparison

To evaluate the clustering performance of different algorithms, we use two widely used measures (Accuracy and NMI (Nasrabadi 2007)). **Normalized Mutual Information (NMI)** (Nasrabadi 2007) is widely used for measuring the performance of clustering algorithms. It can be calculated as $NMI(C, C') = \frac{I(C;C')}{\sqrt{H(C)H(C')}}$, where $H(C)$ and $H(C')$ are the marginal entropies, $I(C; C')$ is the mutual information between C' and C. The **Accuracy** measure, which is calculated as the percentage of target objects going to the

correct cluster, is defined as $Accuracy = \frac{\sum_{i=1}^{k} a_i}{n}$, where a_i is the number of data objects clustered to its corresponding true cluster, k is the number of cluster and n is the number of data objects.

Table 1 shows the clustering accuracy of different competing methods. We can see that the MinSM, LSHSM and SDDS are much more accurate than RGSM. This observation is in agreement with the likelihood plots. On the other hand, the accuracy difference between MinSM, LSHSM and SDDS is negligible. This small difference is due to the mismatch between the likelihood value and clustering accuracy. It should be noted that the difference is small for SDSS and MinSM variants because both achieved the same likelihood value. For the Random Split merge with the worse likelihood, the difference is huge, indicating the clustering results does correlate with likelihood values except for minor variations.

Conclusion

The Split-Merge MCMC (Monte Carlo Markov Chain) is one of the essential and popular variants of MCMC for problems with an unknown number of components. It is a well known that the inference process of SplitMerge MCMC is computational expensive which is not applicable for the large-scale dataset. Existing approaches that try to speed up the split-merge MCMC are stuck in a computational chicken-and-egg loop problem.

In this paper, we proposed MinSM, accelerating Split Merge MCMC via weighted Minhash. The new splitmerge MCMC has constant time update, and at the same time the proposal is informative and needs significantly fewer iterations than random split-merge. Overall, we obtain a sweet tradeoff between convergence and per update cost. Experiments with Gaussian Mixture Model on two real-world datasets demonstrate much faster convergence and better scaling to large datasets.

Acknowledgement

This work was supported by National Science Foundation IIS-1652131, BIGDATA-1838177, RI-1718478, AFOSR-YIP FA9550-18-1-0152, Amazon Research Award, ONR BRC grant on Randomized Numerical Linear Algebra.

References

Andrieu, C.; De Freitas, N.; Doucet, A.; and Jordan, M. I. 2003. An introduction to mcmc for machine learning. *Machine learning* 50(1-2):5–43.

Bengio, S.; Weston, J.; and Grangier, D. 2010. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, 163–171.

Chang, J., and Fisher III, J. W. 2013. Parallel sampling of dp mixture models using sub-cluster splits. In *Advances in Neural Information Processing Systems*, 620–628.

Charikar, M., and Siminelakis, P. 2017. Hashing-based-estimators for kernel density in high dimensions. FOCS.

Chen, B.; Shrivastava, A.; and Steorts, R. C. 2017. Unique entity estimation with application to the syrian conflict. *arXiv preprint arXiv:1710.02690*.

Eronen, L.; Geerts, F.; and Toivonen, H. 2003. A markov chain approach to reconstruction of long haplotypes. In *Biocomputing 2004*. World Scientific. 104–115.

Gionis, A.; Indyk, P.; Motwani, R.; et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, 518–529.

Huelsenbeck, J. P., and Ronquist, F. 2001. Mrbayes: Bayesian inference of phylogenetic trees. *Bioinformatics* 17(8):754–755.

Hughes, M. C.; Fox, E.; and Sudderth, E. B. 2012. Effective split-merge monte carlo methods for nonparametric models of sequential data. In *Advances in neural information processing systems*, 1295–1303.

Jain, S., and Neal, R. M. 2004. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of Computational and Graphical Statistics* 13(1):158–182.

Leskovec, J.; Rajaraman, A.; and Ullman, J. D. 2014. *Mining of massive datasets*. Cambridge university press.

Li, P. 2017. Linearized gmm kernels and normalized random fourier features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 315–324. ACM.

Luo, C., and Shrivastava, A. 2018. Arrays of (locality-sensitive) count estimators (ace): Anomaly detection on the edge. In *Proceedings of the 2018 World Wide Web Conference*, WWW'18, 1439–1448.

Manasse, M.; McSherry, F.; and Talwar, K. 2010. Consistent weighted sampling. *Unpublished technical report) http://research. microsoft. com/en-us/people/manasse* 2.

Medvedovic, M.; Yeung, K. Y.; and Bumgarner, R. E. 2004. Bayesian mixture model based clustering of replicated microarray data. *Bioinformatics* 20(8):1222–1232.

Nasrabadi, N. M. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16(4):049901.

Sharma, A., and Adlakha, N. 2015. A computational model to study the concentrations of dna, mrna and proteins in a growing cell. *Journal of Medical Imaging and Health Informatics* 5(5):945–950.

Shrivastava, A., and Li, P. 2013. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. In *Advances in Neural Information Processing Systems*, 791–799.

Shrivastava, A. 2016. Simple and efficient weighted minwise hashing. In *Advances in Neural Information Processing Systems*, 1498–1506.

Spring, R., and Shrivastava, A. 2017a. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*.

Spring, R., and Shrivastava, A. 2017b. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 445–454. ACM.

Wang, C., and Blei, D. M. 2012. A split-merge mcmc algorithm for the hierarchical dirichlet process. *arXiv preprint arXiv:1201.1657*.

Wang, W., and Russell, S. J. 2015. A smart-dumb/dumb-smart algorithm for efficient split-merge mcmc. In *UAI*, 902–911.