Tree-Guided MCMC Inference for Normalized Random Measure Mixture Models

Juho Lee and Seungjin Choi
Department of Computer Science and Engineering
Pohang University of Science and Technology
77 Cheongam-ro, Nam-gu, Pohang 37673, Korea
{stonecold, seungjin}@postech.ac.kr

Abstract

Normalized random measures (NRMs) provide a broad class of discrete random measures that are often used as priors for Bayesian nonparametric models. Dirichlet process is a well-known example of NRMs. Most of posterior inference methods for NRM mixture models rely on MCMC methods since they are easy to implement and their convergence is well studied. However, MCMC often suffers from slow convergence when the acceptance rate is low. Tree-based inference is an alternative deterministic posterior inference method, where Bayesian hierarchical clustering (BHC) or incremental Bayesian hierarchical clustering (IBHC) have been developed for DP or NRM mixture (NRMM) models, respectively. Although IBHC is a promising method for posterior inference for NRMM models due to its efficiency and applicability to online inference, its convergence is not guaranteed since it uses heuristics that simply selects the best solution after multiple trials are made. In this paper, we present a hybrid inference algorithm for NRMM models, which combines the merits of both MCMC and IBHC. Trees built by IBHC outlines partitions of data, which guides Metropolis-Hastings procedure to employ appropriate proposals. Inheriting the nature of MCMC, our tree-guided MCMC (tgMCMC) is guaranteed to converge, and enjoys the fast convergence thanks to the effective proposals guided by trees. Experiments on both synthetic and real-world datasets demonstrate the benefit of our method.

1 Introduction

Normalized random measures (NRMs) form a broad class of discrete random measures, including Dirichlet process (DP) [1] normalized inverse Gaussian process [2], and normalized generalized Gamma process [3, 4]. NRM mixture (NRMM) model [5] is a representative example where NRM is used as a prior for mixture models. Recently NRMs were extended to dependent NRMs (DNRMs) [6, 7] to model data where exchangeability fails. The posterior analysis for NRM mixture (NRMM) models has been developed [8, 9], yielding simple MCMC methods [10]. As in DP mixture (DPM) models [11], there are two paradigms in the MCMC algorithms for NRMM models: (1) marginal samplers and (2) slice samplers. The marginal samplers simulate the posterior distributions of partitions and cluster parameters given data (or just partitions given data provided that conjugate priors are assumed) by marginalizing out the random measures. The marginal samplers include the Gibbs sampler [10], and the split-merge sampler [12], although it was not formally extended to NRMM models. The slice sampler [13] maintains random measures and explicitly samples the weights and atoms of the random measures. The term "slice" comes from the auxiliary slice variables used to control the number of atoms to be used. The slice sampler is known to mix faster than the marginal Gibbs sampler when applied to complicated DNRM mixture models where the evaluation of marginal distribution is costly [7].
The main drawback of MCMC methods for NRMM models is their poor scalability, due to the nature of MCMC methods. Moreover, since the marginal Gibbs sampler and slice sampler iteratively sample the cluster assignment variable for a single data point at a time, they easily get stuck in local optima. Split-merge sampler may resolve the local optima problem to some extent, but is still problematic for large-scale datasets since the samples proposed by split or merge procedures are rarely accepted. Recently, a deterministic alternative to MCMC algorithms for NRM (or DNRM) mixture models were proposed [14], extending Bayesian hierarchical clustering (BHC) [15] which was developed as a tree-based inference for DP mixture models. The algorithm, referred to as incremental BHC (IBHC) [14] builds binary trees that reflects the hierarchical cluster structures of datasets by evaluating the approximate marginal likelihood of NRMM models, and is well suited for the incremental inferences for large-scale or streaming datasets. The key idea of IBHC is to consider only exponentially many posterior samples (which are represented as binary trees), instead of drawing indefinite number of samples as in MCMC methods. However, IBHC depends on the heuristics that chooses the best trees after the multiple trials, and thus is not guaranteed to converge to the true posterior distributions.

In this paper, we propose a novel MCMC algorithm that elegantly combines IBHC and MCMC methods for NRMM models. Our algorithm, called the tree-guided MCMC, utilizes the trees built from IBHC to proposes a good quality posterior samples efficiently. The trees contain useful information such as dissimilarities between clusters, so the errors in cluster assignments may be detected and corrected with less efforts. Moreover, designed as a MCMC methods, our algorithm is guaranteed to converge to the true posterior, which was not possible for IBHC. We demonstrate the efficiency and accuracy of our algorithm by comparing it to existing MCMC algorithms.

2 Background

Throughout this paper we use the following notations. Denote by \([n] = \{1, \ldots, n\}\) a set of indices and by \(X = \{x_i \mid i \in [n]\}\) a dataset. A partition \(\Pi_{[n]}\) of \([n]\) is a set of disjoint nonempty subsets of \([n]\) whose union is \([n]\). Cluster \(c\) is an entry of \(\Pi_{[n]}\), i.e., \(c \in \Pi_{[n]}\). Data points in cluster \(c\) is denoted by \(X_c = \{x_i \mid i \in c\}\) for \(c \in \Pi_{[n]}\). For the sake of simplicity, we often use \(i\) to represent a singleton \(\{i\}\) for \(i \in [n]\). In this section, we briefly review NRMM models, existing posterior inference methods such as MCMC and IBHC.

2.1 Normalized random measure mixture models

Let \(\mu\) be a homogeneous completely random measure (CRM) on measure space \((\Theta, \mathcal{F})\) with Lévy intensity \(\rho\) and base measure \(H\), written as \(\mu \sim \text{CRM}(\rho H)\). We also assume that,

\[
\int_0^\infty \rho(dw) = \infty, \quad \int_0^\infty (1 - e^{-w})\rho(dw) < \infty,
\]

so that \(\mu\) has infinitely many atoms and the total mass \(\mu(\Theta)\) is finite: \(\mu = \sum_{j=1}^\infty w_j \delta_{\theta_j}\), \(\mu(\Theta) = \sum_{j=1}^\infty w_j < \infty\). A NRM is then formed by normalizing \(\mu\) by its total mass \(\mu(\Theta)\). For each index \(i \in [n]\), we draw the corresponding atoms from NRM, \(\theta_i | \mu \sim \mu/\mu(\Theta)\). Since \(\mu\) is discrete, the set \(\{\theta_i | i \in [n]\}\) naturally form a partition of \([n]\) with respect to the assigned atoms. We write the partition as a set of sets \(\Pi_{[n]}\) whose elements are non-empty and non-overlapping subsets of \([n]\), and the union of the elements is \([n]\). We index the elements (clusters) of \(\Pi_{[n]}\) with the symbol \(c\), and denote the unique atom assigned to \(c\) as \(\theta_c\). Summarizing the set \(\{\theta_i | i \in [n]\}\) as \((\Pi_{[n]}; \{\theta_c | c \in \Pi_{[n]}\})\), the posterior random measure is written as follows:

**Theorem 1.** (\([9]\)) Let \((\Pi_{[n]}; \{\theta_c | c \in \Pi_{[n]}\})\) be samples drawn from \(\mu/\mu(\Theta)\) where \(\mu \sim \text{CRM}(\rho H)\). With an auxiliary variable \(u \sim \text{Gamma}(n, \mu(\Theta))\), the posterior random measure is written as

\[
\mu | u + \sum_{c \in \Pi_{[n]}} w_c \delta_{\theta_c}, \tag{2}
\]

where

\[
\rho_u(dw) := e^{-w} \rho(dw), \quad u | \mu \sim \text{CRM}(\rho_u H), \quad P(dw_c) \propto w_c^{u-1} \rho_u(dw_c). \tag{3}
\]
Moreover, the marginal distribution is written as
\[ P(\Pi_n, \{d \theta_i, c \in \Pi_n\}, du) = \frac{u^n e^{-\psi(u)}}{\Gamma(n)} \prod_{c \in \Pi_n} \kappa_{\rho}(|c|, u) H(d \theta_c), \]
where
\[ \psi(u) := \int_0^\infty (1 - e^{-u \rho}) \rho(d w), \quad \kappa_{\rho}(|c|, u) := \int_0^\infty u^{|c|} \rho_u(dw). \]

Using Equation (4), the predictive distribution for the new atom \( \theta \) is written as
\[ P(d \theta|\{\theta_i\}, u) \propto \kappa_{\rho}(1, u) H(d \theta) + \sum_{c \in \Pi_n} \frac{\kappa_{\rho}(|c| + 1, u)}{\kappa_{\rho}(|c|, u)} \delta_{\theta_c}(d \theta). \]

The most general CRM may be used is the generalized Gamma \([3]\), with Lévy intensity \( \rho(dw) = \frac{\alpha \sigma}{\Gamma(1 - \sigma)} w^{-\sigma - 1} e^{-\sigma w} dw \). In NRMM models, the observed dataset \( X \) is assumed to be generated from a likelihood \( P(dx|\theta) \) with parameters \( \{\theta_i\} \) drawn from NRM. We focus on the conjugate case where \( H \) is conjugate to \( P(dx|\theta) \), so that the integral \( P(dx_i|\theta) := \int_\Theta H(d \theta) \prod_{i \in c} P(dx_i|\theta) \) is tractable.

### 2.2 MCMC Inference for NRMM models

The goal of posterior inference for NRMM models is to compute the posterior \( P(\Pi_n, \{d \theta_i\}, du|X) \) with the marginal likelihood \( P(dX) \).

**Marginal Gibbs Sampler:** marginal Gibbs sampler is based on the predictive distribution (6). At each iteration, cluster assignments for each data point is sampled, where \( x_i \) may join an existing cluster \( c \) with probability proportional to \( \frac{s_x(|c|+1, u)}{\kappa_{\rho}(|c|, u)} P(dx_i|X_c) \), or create a novel cluster with probability proportional to \( \kappa_{\rho}(1, u) P(dx_i) \).

**Slice sampler:** instead of marginalizing out \( \mu \), slice sampler explicitly sample the atoms and weights \( \{w_j, \theta_i^*\} \) of \( \mu \). Since maintaining infinitely many atoms is infeasible, slice variables \( \{s_i\} \) are introduced for each data point, and atoms with masses larger than a threshold (usually set as \( \min_i s_i \) kept and remaining atoms are added on the fly as the threshold changes. At each iteration, \( x_i \) is assigned to the \( j \)th atom with probability \( \mathbb{I}[s_j < w_j] P(dx_i|\theta_i^*) \).

**Split-merge sampler:** both marginal Gibbs and slice sampler alter a single cluster assignment at a time, so are prone to the local optima. Split-merge sampler, originally developed for DPM, is a marginal sampler that is based on (6). At each iteration, instead of changing individual cluster assignments, split-merge sampler splits or merges clusters to propose a new partition. The split or merge is proposed by a procedure called the restricted Gibbs sampling, which is Gibbs sampling restricted to the clusters to split or merge. The proposed partitions are accepted or rejected according to Metropolis-Hastings schemes. Split-merge samplers are reported to mix better than marginal Gibbs sampler.

### 2.3 IBHC Inference for NRMM models

Bayesian hierarchical clustering (BHC, [15]) is a probabilistic model-based agglomerative clustering, where the marginal likelihood of DPM is evaluated to measure the dissimilarity between nodes. Like the traditional agglomerative clustering algorithms, BHC repeatedly merges the pair of nodes with the smallest dissimilarities, and builds binary trees embedding the hierarchical cluster structure of datasets. BHC defines the generative probability of binary trees which is maximized during the construction of the tree, and the generative probability provides a lower bound on the marginal likelihood of DPM. For this reason, BHC is considered to be a posterior inference algorithm for DPM.

Incremental BHC (IBHC, [14]) is an extension of BHC to (dependent) NRMM models. Unlike the original BHC that greedily builds trees, IBHC sequentially insert data points into trees, yielding scalable algorithm that is well suited for online inference. We first explain the generative model of trees, and then explain the sequential algorithm of IBHC.
IBHC aims to maximize the joint probability of the data $X$ and the auxiliary variable $u$:

$$P(dX, du) = \frac{u^{n-1} e^{-\psi(u)} du}{\Gamma(n)} \sum_{\Pi[n]} \prod_{c \in \Pi[n]} \kappa_{\rho}(|c|, u) P(dX_c)$$  \tag{7}$$

Let $t_c$ be a binary tree whose leaf nodes consist of the indices in $c$. Let $l(c)$ and $r(c)$ denote the left and right child of the set $c$ in tree, and thus the corresponding trees are denoted by $t_l(c)$ and $t_r(c)$. The generative probability of trees is described with the potential function $\Gamma$ which is the unnormalized reformulation of the original definition $\Pi$. The potential function of the data $X_c$ given the tree $t_c$ is recursively defined as follows:

$$\phi(X_c|\rho) := \kappa_{\rho}(|c|, u) P(dX_c), \quad \phi(X_c|t_c) = \phi(X_c|\rho) + \phi(X_{l(c)}|t_l(c))\phi(X_{r(c)}|t_r(c)).$$  \tag{8}$$

Here, $\rho$ is the hypothesis that $X_c$ was generated from a single cluster. The first term is proportional to the probability that $\rho$ is true, and came from the term inside the product of (7). The second term is proportional to the probability that $X_c$ was generated from more than two clusters embedded in the subtrees $t_l(c)$ and $t_r(c)$. The posterior probability of $\rho$ is then computed as

$$P(\rho|X_c, t_c) = \frac{1}{1 + d(l(c), r(c))}, \quad \text{where } d(l(c), r(c)) := \frac{\phi(X_{l(c)}|t_l(c))\phi(X_{r(c)}|t_r(c))}{\phi(X_c|\rho)}.$$

$d(\cdot, \cdot)$ is defined to be the dissimilarity between $l(c)$ and $r(c)$. In the greedy construction, the pair of nodes with smallest $d(\cdot, \cdot)$ are merged at each iteration. When the minimum dissimilarity exceeds one ($P(\rho|X_c, t_c) < 0.5$), $\rho$ is concluded to be false and the construction stops. This is an important mechanism of BHC (and IBHC) that naturally selects the proper number of clusters. In the perspective of the posterior inference, this stopping corresponds to selecting the MAP partition that maximizes $P(\Pi[n]|X, u)$. If the tree is built and the potential function is computed for the entire dataset $X$, a lower bound on the joint likelihood (7) is obtained $\Gamma$:

$$\frac{u^{n-1} e^{-\psi(u)} du}{\Gamma(n)} \phi(X|\Pi[n]) \leq P(dX, du).$$  \tag{10}$$

Now we explain the sequential tree construction of IBHC. IBHC constructs a tree in an incremental manner by inserting a new data point into an appropriate position of the existing tree, without computing dissimilarities between every pair of nodes. The procedure, which comprises three steps, is elucidated in Fig. 1.

**Step 1 (left):** Given $\{x_1, \ldots, x_{i-1}\}$, suppose that trees are built by IBHC, yielding to a partition $\Pi[i-1]$. When a new data point $x_i$ arrives, this step assigns $x_i$ to a tree $t_c$ which has the smallest distance, i.e., $\vec{c} = \arg \min_{c \in \Pi[i-1]} d(i, c)$, or create a new tree $t_i$ if $d(i, \vec{c}) > 1$.

**Step 2 (middle):** Suppose that the tree chosen in Step 1 is $t_c$. Then Step 2 determines an appropriate position of $x_i$ when it is inserted into the tree $t_c$, and this is done by the procedure $\text{SeqInsert}(c, i)$. $\text{SeqInsert}(c, i)$ chooses the position of $i$ among three cases (Fig. 1). Case 1 elucidates an option where $x_i$ is placed on the top of the tree $t_c$. Case 2 and 3 show options where $x_i$ is added as a sibling of the subtree $t_l(c)$ or $t_r(c)$, respectively. Among these three cases, the one with the highest potential function $\phi(X_{x_{i+1}}|t_{c,x_{i+1}})$ is selected, which can easily be done by comparing $d(l(c), r(c))$, $d(l(c), i)$ and $d(r(c), i)$ $\Gamma$. If $d(l(c), i)$ is the smallest, then Case 1 is selected and the insertion terminates. Otherwise, if $d(l(c), i)$ is the smallest, $x_i$ is inserted into $t_l(c)$ and $\text{SeqInsert}(l(c), i)$ is recursively executed. The same procedure is applied to the case where $d(r(c), i)$ is smallest.
Step 3 (right): After Step 1 and 2 are applied, the potential functions of $t_{c,i}$ should be computed again, starting from the subtree of $t_c$ to which $x_i$ is inserted, to the root $t_{c,i}$. During this procedure, updated $d(\cdot, \cdot)\text{ values may exceed 1.}$ In such a case, we split the tree at the level where $d(\cdot, \cdot) > 1$, and re-insert all the split nodes.

After having inserted all the data points in $X$, the auxiliary variable $u$ and hyperparameters for $\rho(dw)$ are resampled, and the tree is reconstructed. This procedure is repeated several times and the trees with the highest potential functions are chosen as an output.

3 Main results: A tree-guided MCMC procedure

IBHC should reconstruct trees from the ground whenever $u$ and hyperparameters are resampled, and this is obviously time consuming, and more importantly, converge is not guaranteed. Instead of completely reconstructing trees, we propose to refine the parts of existing trees with MCMC. Our algorithm, called tree-guided MCMC (tgMCMC), is a combination of deterministic tree-based inference and MCMC, where the trees constructed via IBHC guides MCMC to propose good-quality samples. tgMCMC initialize a chain with a single run of IBHC. Given a current partition $\Pi^{t,*}_{[n]}$, and trees $\{t_c | c \in \Pi^{t,*}_{[n]}\}$, tgMCMC proposes a novel partition $\Pi^{t}_{[n]}$ by global and local moves. Global moves split or merges clusters to propose $\Pi^{t}_{[n]}$, and local moves alters cluster assignments of individual data points via Gibbs sampling. We first explain the two key operations used to modify tree structures, and then explain global and local moves. More details on the algorithm can be found in the supplementary material.

3.1 Key operations

SampleSub($c,p$): given a tree $t_c$, draw a subtree $t'_{c'}$ with probability $\propto d(l(c'),r(c')) + \epsilon$. $\epsilon$ is added for leaf nodes whose $d(\cdot, \cdot) = 0$, and set to the maximum $d(\cdot, \cdot)$ among all subtrees of $t_c$. The drawn subtree is likely to contain errors to be corrected by splitting. The probability of drawing $t'_{c'}$ is multiplied to $p$, where $p$ is usually set to transition probabilities.

StocInsert($S,c,p$): a stochastic version of IBHC. $c$ may be inserted to $c' \in S$ via SeqInsert($c',c$) with probability $\frac{d^{-1}(c',c)}{1 + \sum_{c'' \in S} d^{-1}(c'',c)}$, or may just be put into $S$ (create a new cluster in $S$) with probability $\frac{1}{1 + \sum_{c'' \in S} d^{-1}(c'',c)}$. If $c$ is inserted via SeqInsert, the potential functions are updated accordingly, but the trees are not split even if the update dissimilarities exceed 1. As in SampleSub, the probability is multiplied to $p$.

3.2 Global moves

The global moves of tgMCMC are tree-guided analogy to split-merge sampling. In split-merge sampling, a pair of data points are randomly selected, and split partition is proposed if they belong to the same cluster, or merged partition is proposed otherwise. Instead, tgMCMC finds the clusters that are highly likely to be split or merged using the dissimilarities between trees, which goes as follows in detail. First, we randomly pick a tree $t_c$ in uniform. Then, we compute $d(c,c')$ for
of running Gibbs sampling for all data points, we run Gibbs sampling for a subset of data points. In local moves, we resample cluster assignments of individual data points via Gibbs sampling. If a split partition to propose is

\[ \pi \in \Pi \backslash c, \]  

the reverse transition probability is computed as a sum of all those possibilities:

\[ q(\pi^*|\pi) = \sum_{c' \in S} \frac{1}{\Pi^*_{\pi}} \prod_{c' \in \Pi^*_{\pi} \setminus c'} \frac{d(c', c'')I[c'' \notin S]}{1 + d(c', c'')}. \]  

(11)

**Merge case:** suppose that we have \( M = \{c_1, \ldots, c_m\} \)\(^1\) The merged partition to propose is given as \( \Pi^*_{\pi} = (\Pi_{\pi}|M) \cup c_{m+1}, \) where \( c_{m+1} = \bigcup_{i=1}^{m} c_i. \) We construct the corresponding binary tree as a cascading tree, where we put \( c_1, \ldots, c_m \) on top of \( c \) in order (Fig. 2, bottom). To compute the reverse transition probability \( q(\Pi_{\pi}|\Pi^*_{\pi}) \), we should compute the probability of splitting \( c_{m+1} \) back into \( c_1, \ldots, c_m. \) For this, we should first choose \( c_{m+1} \) and put nothing into the set \( M \) to provoke splitting. \( q(\Pi_{\pi}|\Pi^*_{\pi}) \) up to this step is \( \frac{1}{\Pi^*_{\pi}} \prod_{c' \in \Pi^*_{\pi} \setminus c} \frac{d(c_{m+1}, c')}{1 + d(c_{m+1}, c')} \). Then, we should sample the parent of \( c \) (the subtree connecting \( c \) and \( c_1 \) ) via \( \text{SampleSub}(c_{m+1}, q(\Pi_{\pi}|\Pi^*_{\pi})) \), and this would result in \( S = \{c_1, c_1\} \) and \( Q = \{c_2, \ldots, c_m\}. \) Finally, we insert \( c_i \in Q \) into \( S \) via \( \text{StocInsert}(S, c_i, q(\Pi_{\pi}|\Pi^*_{\pi})) \) for \( i = 2, \ldots, m \), where we select each \( c(i) \) to create a new cluster in \( S. \) Corresponding update to \( q(\Pi_{\pi}|\Pi^*_{\pi}) \) by \( \text{StocInsert} \) is

\[ q(\Pi_{\pi}|\Pi^*_{\pi}) \leftarrow q(\Pi_{\pi}|\Pi^*_{\pi}) \frac{1}{1 + d^{-1}(c, c_1) + \sum_{j=1}^{m-1} d^{-1}(c_j, c_1)}. \]  

(12)

Once we’ve proposed \( \Pi^*_{\pi} \) and computed both \( q(\Pi^*_{\pi}|\Pi_{\pi}) \) and \( q(\Pi_{\pi}|\Pi^*_{\pi}), \) \( \Pi^*_{\pi} \) is accepted with probability \( \min\{1, r\} \) where \( r = \frac{\text{p}(dX, du, \Pi^*_{\pi} | \Pi_{\pi}) q(\Pi_{\pi}|\Pi^*_{\pi})}{\text{p}(dX, du, \Pi_{\pi} | \Pi_{\pi}^*).} \)

**Ergodicity of the global moves:** to show that the global moves are ergodic, it is enough to show that we can move an arbitrary point \( i \) from its current cluster \( c \) to any other cluster \( c' \) in finite step. This can easily be done by a single split and merge moves, so the global moves are ergodic.

**Time complexity of the global moves:** the time complexity of \( \text{StocInsert}(S, c, p) \) is \( O(|S| + h), \) where \( h \) is a height of the tree to insert \( c. \) The total time complexity of split proposal is mainly determined by the time to execute \( \text{StocInsert}(S, c, p). \) This procedure is usually efficient, especially when the trees are well balanced. The time complexity to propose merged partition is \( O(|\Pi_{\pi}| + M). \)

### 3.3 Local moves

In local moves, we resample cluster assignments of individual data points via Gibbs sampling. If a leaf node \( i \) is moved from \( c \) to \( c', \) we detach \( i \) from \( t_c \) and run \( \text{SeqInsert}(c', i) \)\(^2\). Here, instead of running Gibbs sampling for all data points, we run Gibbs sampling for a subset of data points \( S, \) which is formed as follows. For each \( c \in \Pi_{\pi}, \) we draw a subtree \( t_c \) by \( \text{SampleSub}. \) Then, we

\(^1\)Here, we restrict \( \text{SampleSub} \) to sample non-leaf nodes, since leaf nodes cannot be split.

\(^2\)We assume that clusters are given their own indices (such as hash values) so that they can be ordered.

\(^3\)We do not split even if the update dissimilarity exceed one, as in \( \text{StocInsert}. \)
4 Experiments

In this section, we compare marginal Gibbs sampler (Gibbs), split-merge sampler (SM) and tgMCMC on synthetic and real datasets.

4.1 Toy dataset

We first compared the samplers on simple toy dataset that has 1,300 two-dimensional points with 13 clusters, sampled from the mixture of Gaussians with predefined means and covariances. Since the partition found by IBHC is almost perfect for this simple data, instead of initializing with IBHC, we initialized the binary tree (and partition) as follows. As in IBHC, we sequentially inserted data we initialized the binary tree (and partition) as follows. As in IBHC, we sequentially inserted data.

The average log value of the acceptance ratio $r$ is not very sensitive to the parameter, so we can control the tradeoff between iteration time and mixing rates.

**Cycling:** at each iteration of tgMCMC, we cycle the global moves and local moves, as in split-merge sampling. We first run the global moves for $G$ times, and run a single sweep of local moves. Setting $G = 20$ and $D = 2$ were the moderate choice for all data we’ve tested.

Figure 3: Experimental results on toy dataset. (Top row) scatter plot of toy dataset, log-likelihoods of three samplers with DP, log-likelihoods with NGGP, log-likelihoods of tgMCMC with varying $G$ and varying $D$. (Bottom row) The statistics of three samplers with DP and the statistics of three samplers with NGGP.

Figure 4: Average log-likelihood plot and the statistics of the samplers for 10K dataset.

\[ H(d\mu, d\Lambda) = N(d\mu|m, (r\Lambda)^{-1})W(d\Lambda|\Psi^{-1}, \nu), \]

where $r = 0.1, \nu = d + 6, d$ is the dimensionality, $m$ is the sample mean and $\Psi = \Sigma/(10 \cdot \det(\Sigma))^{1/d}$ ($\Sigma$ is the sample covariance). We compared the samplers using both DP and NGGP priors. For tgMCMC, we fixed the number of global moves $G = 20$ and the parameter for local moves $D = 2$, except for the cases where we controlled them explicitly. All the samplers were run for 10 seconds, and repeated 10 times. We compared the joint log-likelihood $\log p(dX, \Pi, \nu, du)$ of samples and the effective sample size (ESS) of the number of clusters found. For SM and tgMCMC, we compared the average log value of the acceptance ratio $r$. The results are summarized in Fig. [3] As shown in
the log-likelihood trace plot, tgMCMC quickly converged to the ground truth solution for both DP and NGGP cases. Also, tgMCMC mixed better than other two samplers in terms of ESS. Comparing the average log $r$ values of SM and tgMCMC, we can see that the partitions proposed by tgMCMC is more often accepted. We also controlled the parameter $G$ and $D$; as expected, higher $G$ resulted in faster convergence. However, smaller $D$ (more data points involved in local moves) did not necessarily mean faster convergence.

4.2 Large-scale synthetic dataset

We also compared the three samplers on larger dataset containing 10,000 points, which we will call 10K dataset, generated from six-dimensional mixture of Gaussians with labels drawn from $PY(3, 0.8)$. We used the same base measure and initialization with those of the toy datasets, and used the NGGP prior. We ran the samplers for 1,000 seconds and repeated 10 times. Gibbs and SM were too slow, so the number of samples produced in 1,000 seconds were too small. Hence, we also compared Gibbs_sub and SM_sub, where we uniformly sampled the subset of data points and ran Gibbs sweep only for those sampled points. We controlled the subset size to make their running time similar to that of tgMCMC. The results are summarized in Fig. 4. Again, tgMCMC outperformed other samplers both in terms of the log-likelihoods and ESS. Interestingly, SM was even worse than Gibbs, since most of the samples proposed by split or merge proposal were rejected. Gibbs_sub and SM_sub were better than Gibbs and SM, but still failed to reach the best state found by tgMCMC.

4.3 NIPS corpus

We also compared the samplers on NIPS corpus containing 1,500 documents with 12,419 words. We used the multinomial likelihood and symmetric Dirichlet base measure $Dir(0.1)$, used NGGP prior, and initialized the samplers with normal IBHC. As for the 10K dataset, we compared Gibbs_sub and SM_sub along. We ran the samplers for 10,000 seconds and repeated 10 times. The results are summarized in Fig. 5. tgMCMC outperformed other samplers in terms of the log-likelihood; all the other samplers were trapped in local optima and failed to reach the states found by tgMCMC. However, ESS for tgMCMC were the lowest, meaning the poor mixing rates. We still argue that tgMCMC is a better option for this dataset, since we think that finding the better log-likelihood states is more important than mixing rates.

5 Conclusion

In this paper we have presented a novel inference algorithm for NRMM models. Our sampler, called tgMCMC, utilized the binary trees constructed by IBHC to propose good quality samples. tgMCMC explored the space of partitions via global and local moves which were guided by the potential functions of trees. tgMCMC was demonstrated to be outperform existing samplers in both synthetic and real world datasets.

Acknowledgments: This work was supported by the IT R&D Program of MSIP/IITP (B0101-15-0307, Machine Learning Center), National Research Foundation (NRF) of Korea (NRF-2013R1A2A2A01067464), and IITP-MSRA Creative ICT/SW Research Project.

\ref{https://archive.ics.uci.edu/ml/datasets/Bag+of+Words}
A Key operations

We first explain two key operations used for tgMCMC.

SampleSub(c, p): given a tree tc, SampleSub(c, p) aims to draw a subtree tc′ of tc. We want tc′ to have high dissimilarity between its subtrees, so we draw tc′ with probability proportional to d(l(c)c′, r(c)c′) + ϵ, where ϵ is the maximum dissimilarity between subtrees. ϵ is added for leaf nodes, whose dissimilarity between subtrees is defined to be zero. See Figure 6.

\[ p_t = \frac{\sum_{j = 1}^3 \{d(l(c_j), r(c_j)) + \epsilon\}}{1 + \sum_{i = 1}^3 d^{-1}(c_i, c)} \]

Figure 6: An illustrative example for SampleSub operation.

As depicted in Figure 6, the probability of drawing the subtree tc, is computed as pt. If we draw tc as a result, corresponding probability pt is multiplied into p; p ← p · pt. This procedure is needed to compute the forward transition probability. If SampleSub takes another argument as SampleSub(c, c′, p), we don’t do the actual sampling but compute the probability of drawing subtree tc′ from tc and multiply the probability to p. In above example, SampleSub(c, c′, p) does p ← p · pt. This operation is needed to compute the reverse transition probability.

StocInsert(S, c, p): given a set of clusters S (and corresponding trees), StocInsert inserts c into one of the trees tc ∈ S, or just put c in S without insertion. If tc is inserted into tc′, the insertion is done by SeqInsert(c, c′). See Figure 7 for example. The set S contains three clusters, and tc may be inserted into c ∈ S via SeqInsert(c, c) with probability pt, or c may just be put into S without insertion with probability p∗. In the original IBHC algorithm, tc is inserted into the tree c with smallest d(c, c), or put into S without insertion if the minimum d(c, c) exceeds one. StocInsert is a probabilistic operation of this procedure, where the probability pt is proportional to d−1(c, c). As emphasized in the paper, the trees are not split after the update of the dissimilarities of inserted trees; this is to ensure reversibility and make the computation of transition probability easier. Even if some dissimilarity exceeds one after the insertion, we expect StocInsert operation to find those nodes needed to be split. After the insertion, corresponding probability is multiplied to p, as in StocInsert operation. If StocInsert takes another arguments as StocInsert(S, c, c′, p), we don’t do insertion but compute the probability of inserting c into c′ ∈ S and multiply it to p. In above example, StocInsert(S, c, c′, p) multiplies p ← p · pt. StocInsert(S, c, ∅, p) multiplies the probability of creating a new tree with c, p ← p · p∗.

B Global moves

In this section, we explain global moves for our tgMCMC.
B.1 Selection between splitting and merging

In global moves, we randomly choose between splitting and merging operations. The basic principle is this: pick a tree, and find candidate trees that can be merged with the picked tree. If there is any candidate, invoke merge operation, and invoke split operation otherwise.

As an example, suppose that our current partition is $\Pi_{[n]} = \{c_i\}_{i=1}^6$, with 6 clusters. We first uniformly pick a cluster among them, and initialize the forward transition probability $q(\Pi^n_{[n]}|\Pi_{[n]}) = 1/6$. Suppose that $c_1$ is selected. Then, for $\{c_i\}_{i=2}^6$, we put $c_i$ into a set $M$ with probability $(1 + d(c_1, c_i))^{-1}$; note that this probability is equal to $p(h_{c_1 \cup c_i}|X(c_i \cup c_1))$, the probability of merging $c_1$ and $c_i$. The splitting is invoked if $M$ contains no cluster, and merging is invoked otherwise.

B.2 Splitting

Suppose that $M$ contains no cluster. The forward transition probability up to this is

$$q(\Pi^n_{[n]}|\Pi_{[n]}) = \frac{1}{6} \prod_{i=2}^{6} \frac{d(c_1, c_i)}{1 + d(c_1, c_i)}.$$

Now suppose that $c_1$ looks like in Figure 8. The split proposal $\Pi^n_{[n]}$ is then proposed according to the following procedure. First, a subtree $c^*$ of $c_1$ is sampled via $\text{SampleSub}$ procedure. Then, the tree is cut at $c^*$, collecting $S = \{l(c^*), r(c^*)\}$ and remaining split nodes $Q = \{c_7, c_8\}$. Nodes in $Q$ are inserted into $S$ via $\text{StocInsert}$. Since $(l(c^*), r(c^*)\}$ were initialized to be split in $S$, the resulting partition must have more than two clusters. During $\text{SampleSub}$ and $\text{StocInsert}$, every intermediate transition probabilities are multiplied into $q(\Pi^n_{[n]}|\Pi_{[n]})$. The final partition $\Pi^n_{[n]}$ to propose is $\{c_i\}_{i=2}^6 \cup \{c_9, c_{10}\}$.

B.3 Merging

Suppose that $M$ contains three clusters; $M = \{c_2, c_3, c_4\}$. The forward transition probability up to this is

$$q(\Pi^n_{[n]}|\Pi_{[n]}) = \frac{1}{6} \prod_{i=2}^{4} \frac{1}{1 + d(c_1, c_i)} \prod_{i=5}^{6} \frac{d(c_1, c_i)}{1 + d(c_1, c_i)}.$$

In partition space, there is only one way to merge $c_1$ and $M = \{c_2, c_3, c_4\}$ into a single cluster $c_9 = c_1 \cup c_2 \cup c_3 \cup c_4$, so no transition probability is multiplied. However, there can be many trees representing $c_9$, since we can merge the nodes in any order or topology, in terms of tree. Hence, we fix the merged tree to be a cascaded tree, merged in order of indices of nodes; see Figure 9. As we wrote in our paper, we assume that each nodes are given unique indices (such as hash values) so
that they can be sorted in order. In this example, the indices for $c_1$ and $c_2$ are 1 and 2, respectively. Then, we build a cascaded tree, merged in order $c_1, c_2, c_3, c_4$. The resulting merged partition is then $\Pi_{[n]}^* = \{c_5, c_6, c_9\}$.

### B.4 Computing reverse transition probability for splitting

Now we explain how to compute the reverse transition probability $q(\Pi_{[n]}|\Pi_{[n]}^*)$ for splitting case. We start from the illustrative partition $\Pi_{[n]}^* = \{c_5, c_6, c_9\}$ in subsection B.2. Starting from $\Pi_{[n]}^*$, we must merge $c_9$ and $c_{10}$ back into $c_1 = c_9 \cup c_{10}$ to retrieve $\Pi_{[n]}$. For this, there are two possibilities: we first pick $c_9$ and collect $M = \{c_{10}\}$, or pick $c_{10}$ and collect $M = \{c_9\}$. Hence, the reverse transition probability is computed as

$$q(\Pi_{[n]}|\Pi_{[n]}^*) = \frac{1}{7} \left( \sum_{i=1}^{5} \frac{d(c_9, c_i)}{1 + d(c_9, c_i)} + \frac{1}{1 + d(c_9, c_{10})} \right) + \frac{1}{7} \left( \sum_{i=1}^{5} \frac{d(c_{10}, c_i)}{1 + d(c_{10}, c_i)} + \frac{1}{1 + d(c_{10}, c_9)} \right).$$

As we explained in subsection B.3, no more reverse transition probabilities are needed since there is only one way to merge nodes back into $c_1$ in partition space.

### B.5 Computing reverse transition probability for merging

We explain how to compute the reverse transition probability for merging case, starting from the illustrative partition in subsection B.3 $\Pi_{[n]}^* = \{c_5, c_6, c_9\}$. See Figure 9. To retrieve $\Pi_{[n]} = \{c_1\}^{\delta}_{i=1}$, we should split $c_9$ into $c_1, c_2, c_3, c_4$. For this, we should first invoke split operation for $c_9$; pick $c_9$ and collect $M = \emptyset$. The reverse transition probability up to this is

$$q(\Pi_{[n]}|\Pi_{[n]}^*) = \frac{1}{3} \left( \frac{d(c_9, c_5)}{1 + d(c_9, c_5)} + \frac{d(c_9, c_6)}{1 + d(c_9, c_6)} \right).$$

Now, we should split $c_9$ to retrieve the original partition. The achieve this, we should first pick direct parent of $c_1$ and $c_2$ (c7 in Figure 9) via SampleSub. The reverse transition probability for this is computed by SampleSub($c_9, c_7, q(\Pi_{[n]}|\Pi_{[n]}^*)$), and we have $S = \{c_1, c_2\}$ and $Q = \{c_3, c_4\}$. To get the original partition, $c_3$ and $c_4$ should be put into $S$ without insertion, and the reverse transition probability for this is computed by StocInsert($S, c_3, \emptyset, q(\Pi_{[n]}|\Pi_{[n]}^*)$) and StocInsert($S, c_4, \emptyset, q(\Pi_{[n]}|\Pi_{[n]}^*)$).

### C Local moves

Local moves resample cluster assignments of individual data points (leaf nodes) with Gibbs sampling. However, instead of running Gibbs sampling for full data, we select a random subset $S$ and run Gibbs sampling for data points in $S$. $S$ is constructed as follows. Given a partition $\Pi_{[n]}$, for each $c \in \Pi_{[n]}$, we sample its subtree via SampleSub. Then we sample a subtree of drawn subtree again with SampleSub. This procedure is repeated for $D$ times, where $D$ is a parameter set by users. Figure 10 depicts the subsampling procedure for $c \in \Pi_{[n]}$ with $D = 2$. As a result of Figure 10, the leaf nodes $\{i, j, k\}$ are added to $S$. After having collected $S$ for all $c \in \Pi_{[n]}$, we run Gibbs sampling.
for those leaf nodes in $S$, and if a leaf node $i$ should be moved from $c$ to another $c'$, we insert $i$ into $c'$ with $\text{SeqInsert}(c', i)$. We can control the tradeoff between mixing rate and speed by controlling $D$; higher $D$ means more elements in $S$, and thus more data points are moved by Gibbs sampling.

References

[1] T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230, 1973.

[2] A. Lijoi, R. H. Mena, and I. Prünster. Hierarchical mixture modeling with normalized inverse-Gaussian priors. *Journal of the American Statistical Association*, 100(472):1278–1291, 2005.

[3] A. Brix. Generalized Gamma measures and shot-noise Cox processes. *Advances in Applied Probability*, 31:929–953, 1999.

[4] A. Lijoi, R. H. Mena, and I. Prünster. Controlling the reinforcement in Bayesian nonparametric mixture models. *Journal of the Royal Statistical Society B*, 69:715–740, 2007.

[5] E. Regazzini, A. Lijoi, and I. Prünster. Distributional results for means of normalized random measures with independent increments. *The Annals of Statistics*, 31(2):560–585, 2003.

[6] C. Chen, N. Ding, and W. Buntine. Dependent hierarchical normalized random measures for dynamic topic modeling. In *Proceedings of the International Conference on Machine Learning (ICML)*, Edinburgh, UK, 2012.

[7] C. Chen, V. Rao, W. Buntine, and Y. W. Teh. Dependent normalized random measures. In *Proceedings of the International Conference on Machine Learning (ICML)*, Atlanta, Georgia, USA, 2013.

[8] L. F. James. Bayesian Poisson process partition calculus with an application to Bayesian Lévy moving averages. *The Annals of Statistics*, 33(4):1771–1799, 2005.

[9] L. F. James, A. Lijoi, and I. Prünster. Posterior analysis for normalized random measures with independent increments. *Scandinavian Journal of Statistics*, 36(1):76–97, 2009.

[10] S. Favaro and Y. W. Teh. MCMC for normalized random measure mixture models. *Statistical Science*, 28(3):335–359, 2013.

[11] C. E. Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The Annals of Statistics*, 2(6):1152–1174, 1974.

[12] S. Jain and R. M. Neal. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13:158–182, 2000.

[13] J. E. Griffin and S. G. Walker. Posterior simulation of normalized random measure mixtures. *Journal of Computational and Graphical Statistics*, 20(1):241–259, 2011.

[14] J. Lee and S. Choi. Incremental tree-based inference with dependent normalized random measures. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Reykjavik, Iceland, 2014.

[15] K. A. Heller and Z. Ghahrahmani. Bayesian hierarchical clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.