Scalable MCMC for Mixed Membership Stochastic Blockmodels

Wenzhe Li* University of Southern California

Sungjin Ahn* University of Montreal

Max Welling University of Amsterdam

Abstract

We propose a stochastic gradient Markov chain Monte Carlo (SG-MCMC) algorithm for scalable inference in mixed-membership stochastic blockmodels (MMSB). Our algorithm is based on the stochastic gradient Riemannian Langevin sampler and achieves both faster speed and higher accuracy *at every iteration* than the current state-of-the-art algorithm based on stochastic variational inference. In addition we develop an approximation that can handle models that entertain a very large number of communities. The experimental results show that SG-MCMC strictly dominates competing algorithms in all cases.

1 Introduction

Probabilistic graphical models represent a convenient paradigm for modeling complex relationships between a potentially very large number of random variables. Bayesian graphical models [13], where we define priors and infer posteriors over parameters also allow us to quantify model uncertainty and facilitate model selection and averaging. But an increasingly urgent question is whether these models and their inference procedures will be up to the challenge of handling very large "big data" problems.

A large subclass of Bayesian graphical models is represented by so called "topic models" such as latent Dirichlet allocation [4]. For these types of models very efficient inference algorithms have recently been developed, either based on stochastic variational Bayesian inference (SVB) [7,12] or on stochastic gradient Markov chain Monte Carlo (SG-MCMC) [2,3,5,6,19]. Both methods have the important property that they only require a small subset of the data-items for every iteration. In other words, they can be applied to (infinite) streaming data.

An important class of "big data" problems are given by networks. Large networks such as social networks easily run into billions of edges and tens of millions of nodes. An interesting problem in this area is the discovery of communities: densely connected groups of nodes that are only sparsely connected to the rest of the network. Large networks may contain millions of such communities. To model overlapping communities the mixed membership stochastic blockmodel (MMSB) was introduced in [8]. Very recently, an efficient stochastic variational inference algorithm was developed for a special case, the assortative MMSB (a-MMSB) [1], greatly extending the reach of Bayesian posterior inference to realistic large scale problem settings. Inspired by this work, and earlier comparisons between SVB and SG-MCMC on LDA [2] we developed a scalable SG-MCMC algorithm for a-MMSB and compared it against SVB on the community detection problem.

Our conclusion is consistent with the findings of [2], namely that SG-MCMC is also both faster and more accurate than SVB algorithms in this domain. While one should expect SG-MCMC to be more accurate than SVB asymptotically (SVB is asymptotically biased while SG-MCMC is not), it is interesting to observe that SG-MCMC dominates SVB across all iterations, despite the fact that SG-MCMC should have a larger variance contribution to the error.

2 Assortative Mixed-Membership Stochastic Blockmodels

Assortative mixed-membership stochastic blockmodel (a-MMSB) [1] is a special case of MMSB [8] that models the group-structure in a network of N nodes. In particular, each node a in the node set $\mathcal{V}^*$ has a K-dimension probability distribution π_a of participating in the K members of the community set $\mathcal{K}$. For every

*Equal contribution. Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

possible peer b in the network, each node a randomly draws a community z_{ab}. If a pair of nodes (a, b) in the edge set $\mathcal{E}^*$ are in the same community: $z_{ab} = z_{ba} = k$, then they have a significant probability β_k to connect, i.e., $y_{ab} = 1$. Otherwise this probability is small. Each community has its connection strength $\beta_k \in (0, 1)$ which explains how likely its members are linked to each other.

The generative process of a-MMSB is then given by,

1. For each community k, draw community strength $\beta_k \sim \text{Beta}(\eta)$
2. For each node a, draw community memberships $\pi_a \sim \text{Dirichlet}(\alpha)$
3. For each pair of nodes a and b,
   (a) Draw interaction indicator $z_{ab} \sim \pi_a$
   (b) Draw interaction indicator $z_{ba} \sim \pi_b$
   (c) Draw link $y_{ab} \sim \text{Bernoulli}(r)$, where $r = \beta_k$ if $z_{ab} = z_{ba} = k$, and $r = \delta$ otherwise.

Unlike the a-MMSB, the original MMSB maintains pair-wise community strength $\beta_{k,k'}$ for all pairs of the communities. Note that it is trivial to extend the results that we obtain in this paper to the general MMSB model. The joint probability of the above process can be written as:

$$p(y, z, \pi, \beta|\alpha, \eta) = \prod_{a=1}^{N} \prod_{b>a}^{N} p(y_{ab}|z_{ab}, z_{ba}, \beta) p(z_{ab}|\pi_a) p(z_{ba}|\pi_b) \prod_{a=1}^{N} p(\pi_a|\alpha) \prod_{k=1}^{K} p(\beta_k|\eta). \quad (1)$$

Both variational inference [1,4,16] and collapsed Gibbs sampling algorithms [11] have been used successfully for small to medium scale problems. However, the $\mathcal{O}(N^2)$ computational complexity per update prevents it from being applied to large scale networks. A stochastic variational algorithm was developed in [1] to address this issue, where each update only depends on a small mini-batch of the nodes in the network.

3 Stochastic Gradient MCMC Algorithms

Our algorithm will be based on the stochastic gradient Langevin dynamics (SGLD) [3]. To sample from a posterior distribution $p(\theta|\mathcal{X}) \propto p(\mathcal{X}|\theta)p(\theta)$ given N i.i.d. data points $\mathcal{X} = \{x_i\}_{i=1}^N$, SGLD applies the following update rule:

$$\theta^* \leftarrow \theta + \frac{\epsilon_t}{2} \left(\nabla_\theta \log p(\theta_t) + N \bar{g}(\theta; \mathcal{D}_n) \right) + \xi, \quad (2)$$

where $\xi \sim \mathcal{N}(0, \epsilon_t)$ with ϵ_t the step size, $\mathcal{D}_n$ a mini-batch of size n sampled from $\mathcal{X}$, and $\bar{g}(\theta; \mathcal{D}_n) = \frac{1}{|\mathcal{D}_n|} \sum_{x \in \mathcal{D}_n} \nabla_\theta \log p(x|\theta)$. As the step size goes to zero by a schedule satisfying $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$, SGLD samples from the true posterior distribution. In SGLD, the Metropolis-Hastings (MH) accept-reject tests are ignored since the rejection probability goes to zero as the step size collapses to zero. While for a finite step size this results in some bias, the overall error is reduced by the reduction of variance due to the ability to draw many more samples per unit time.

SGLD originated from the Langevin Monte Carlo (LMC) [15] where, unlike SGLD, the gradient is computed exactly using all data points and then a Metropolis-Hastings accept-reject test is applied. Because at each iteration SGLD requires to process only a mini-batch $\mathcal{D}_n$ and ignores the MH test, the computation complexity per iteration is only $\mathcal{O}(n)$ as opposed to $\mathcal{O}(N)$ of LMC. Any mini-batch sampling algorithm in the form of Eqn. (2) is called valid SGLD as long as it guarantees the gradient estimator to be unbiased, i.e., $\mathbb{E}_{\mathcal{D}_n} [N \bar{g}(\theta; \mathcal{D}_n)] = \nabla_\theta \log p(\mathcal{X}|\theta)$ and the variance to be finite [5].

The stochastic gradient Riemannian Langevin dynamics (SGRLD) [2] is a subclass of SGLD which is developed to sample from the probability simplex. By applying Riemannian geometry [15] and using the mini-batch estimator in Eqn. 2, it achieved the state-of-the-art performance for latent Dirichlet allocation (LDA). In particular, for a K-dimensional probability simplex π, it uses the *expanded-mean* re-parameterization trick, where the probability of a category k is given by $\pi_k = \theta_k / \sum_{j=1}^{K} \theta_j$ with $\theta_k \sim \text{Gamma}(\alpha, 1)$ and α a hyperparameter of the Dirichlet distribution $p(\pi|\alpha)$. Then, the update rule becomes

$$\theta_k^* \leftarrow \left| \theta_k + \frac{\epsilon}{2} \left(\alpha - \theta_k + \frac{N}{|\mathcal{D}_n|} \sum_{d \in \mathcal{D}_n} g_d(\theta_k) \right) + (\theta_k)^{\frac{1}{2}} \xi \right|. \quad (3)$$

here $g_d(\theta_k)$ is the gradient of the log posterior w.r.t. θ_k on a data point $d \in \mathcal{D}_n$.

4 Scalable MCMC for a-MMSB

Our algorithm iterates updating local parameters π and a global parameter β. Because both parameters lie on the probability simplex, we start from the SGRLD and modify it to be more efficient. Also, we introduce parameters ϕ and θ to re-parameterize π and β respectively. Then, we alternatively sample in the ϕ and θ spaces, and obtain π and β by normalizing ϕ and θ. From Eqn. 1, summing over the latent variable z, we

obtain the following joint probability,

$$p(y,\pi,\beta|\alpha,\eta) = \prod_a p(\pi_a|\alpha)\prod_k p(\beta_k|\eta)\times \prod_a\prod_{b>a}\sum_{z_{ab},z_{ba}} p(y_{ab},z_{ab},z_{ba}|\beta,\pi_a,\pi_b). \quad (4)$$

4.1 Sampling the global parameter

By the re-parameterization, we have $\beta_k = \theta_{k1}/(\theta_{k0}+\theta_{k1})$, where $\theta_{ki} \sim \text{Gamma}(\eta) \propto \theta_{ki}^{\eta-1}e^{-\theta_{ki}}$. Because $p(y,\pi,\beta|\alpha,\eta)$ decomposes into $p(y,\beta|\pi,\eta)p(\pi|\alpha)$, replacing β by θ, we compute the derivative of the logarithm of Eqn. 4 w.r.t. θ_{ki} for $i=\{0,1\}$ as follows:

$$\frac{\partial \ln p(y,\theta|\pi,\eta)}{\partial\theta_{ki}} = \frac{\partial}{\partial\theta_{ki}}\ln p(\theta_{ki}|\eta) + \sum_a\sum_{b>a} g_{ab}(\theta_{ki}), \quad (5)$$

where $g_{ab}(\theta_{ki}) = \frac{\partial}{\partial\theta_{ki}}\ln\sum_{z_{ab},z_{ba}} p(y_{ab},z_{ab},z_{ba}|\theta,\pi_a,\pi_b)$ which, similar to SGRLD for LDA [2], we can rewrite as

$$g_{ab}(\theta_{ki}) = \mathbb{E}\left[\mathbb{I}[z_{ab}=z_{ba}=k]\left(\frac{|1-i-y_{ab}|}{\theta_{ki}} - \frac{1}{\theta_k}\right)\right]. \quad (6)$$

where $\theta_k = \sum_i \theta_{ki}$ and $\mathbb{I}[S]$ is equal to 1 if a condition S is TRUE and 0 otherwise. The expectation is w.r.t. the posterior distribution of latent variables z_{ab} and z_{ba},

$$p(z_{ab}=k, z_{ba}=l|y_{ab},\pi_a,\pi_b,\beta) \quad (7)$$
$$\propto f_{ab}^{(y)}(k,l) = \begin{cases}\beta_k^y(1-\beta_k)^{(1-y)}\pi_{ak}\pi_{bk}, & \text{if } k=l\\ \delta^y(1-\delta)^{(1-y)}\pi_{ak}\pi_{bl}, & \text{if } k\neq l\end{cases}$$

here we used simple notation y instead of y_{ab}. Unlike the SGRLD for LDA [2], we compute the expectation in Eqn. (6) analytically by computing the normalization constant $Z_{ab}^{(y)} = \sum_{k=1}^K\sum_{l=1}^K f_{ab}^{(y)}(k,l)$ which can be reduced to $\mathcal{O}(K)$ computation as follows

$$Z_{ab}^{(y)} = \delta^y(1-\delta)^{(1-y)} + \sum_{k=1}^K\left(\beta_k^y(1-\beta_k)^{(1-y)} - \delta^y(1-\delta)^{(1-y)}\right)\pi_{ak}\pi_{bk} \quad (8)$$

Then Eqn. (6) becomes

$$g_{ab}(\theta_{ki}) = \frac{f_{ab}^{(y)}(k,k)}{Z_{ab}^{(y)}}\left(\frac{|1-i-y|}{\theta_{ki}} - \frac{1}{\theta_k}\right). \quad (9)$$

Plugging this into Eqn. 3, we obtain the update rule for the global parameter,

$$\theta_{ki}^* \leftarrow \left|\theta_{ki} + \frac{\epsilon}{2}\left\{\eta - \theta_{ki} + h(\mathcal{E}_{n_t})\sum_{(a,b)\in\mathcal{E}_{n_t}} g_{ab}(\theta_{ki})\right\} + (\theta_{ki})^{\frac{1}{2}}\xi_{ki}\right|, \quad (10)$$

here $\mathcal{E}_{n_t}$ is a mini-batch of n_t node pairs sampled from $\mathcal{E}^*$ for which we use the following strategy.

Stratified sampling: considering that the number of links is much smaller than that of non-links, we can reduce the variance of the gradient using stratified sampling, similar to the method used in [1]. For this, at every iteration we first randomly select a node a and then toss a coin with probability 0.5 to decide whether to sample link edges or non-link edges for node a. If it is a link, we assign all of the link edges of node a to $\mathcal{E}_{n_t}$. Otherwise, i.e. if it is non-link, we uniformly sample a mini-batch of N/m non-link edges from the entire set of non-link edges and assign it to $\mathcal{E}_{n_t}$. Here, the m is a hyper-parameter. Note that the size of $|\mathcal{E}_{n_t}|$ will thus be much smaller than the total number of $N(N-1)/2$ edges when m is reasonably large. Then, to ensure that the gradient is unbiased, a *scaling parameter* $h(\mathcal{E}_{n_t})$ is multiplied. Specifically, $h(\mathcal{E}_{n_t})$ is set to N when $\mathcal{E}_{n_t}$ is a set of link edges and to mN otherwise.

Because the global parameters $\{\beta_k\}$ does not change very fast compared to the local parameters $\{\pi_{ak}\}$, in practice we update only a random subset of the $\{\beta_k\}$ at each iteration.

4.2 Sampling the local parameters

Similar to the global parameter, we re-parameterize the local parameter π_a such that $\pi_{ak} = \phi_{ak}/\sum_{j=1}^K\phi_{aj}$, with $\phi_{ak} \sim \text{Gamma}(\alpha) \propto \phi_{ak}^{\alpha-1}e^{-\phi_{ak}}$. Then, taking the derivative of the log of Eqn. 4 w.r.t. ϕ_{ak}, we obtain

$$\frac{\partial \ln p(y,\phi|\beta,\alpha)}{\partial\phi_{ak}} = \frac{\partial}{\partial\phi_{ak}}\ln p(\phi_{ak}|\alpha) + \sum_b g_{ab}(\phi_{ak}) \quad (11)$$

where $g_{ab}(\phi_{ak}) = \frac{\partial}{\partial\phi_{ak}}\ln\sum_{z_{ab},z_{ba}} p(y_{ab},z_{ab},z_{ba}|\beta,\phi_a,\phi_b)$ which can be written as

$$g_{ab}(\phi_{ak}) = \mathbb{E}\left[\frac{\mathbb{I}[z_{ab}=k]}{\phi_{ak}} - \frac{1}{\phi_{a\cdot}}\right]. \quad (12)$$

Here the expectation is w.r.t. the distribution in Eqn. (8). To compute the expectation analytically, we first integrate out z_{ba} from Eqn. (8) because the expectation depends only on z_{ab}, and obtain the following probability up to a normalization constant

$$f_{ab}^{(y)}(k) = \sum_{l=1}^K f_{ab}^{(y)}(k,l) = \pi_{ak}\left\{\beta_k^y(1-\beta_k)^{(1-y)}\pi_{bk} + \delta^y(1-\delta)^{(1-y)}(1-\pi_{bk})\right\}. \quad (13)$$

Then we obtain the normalization term by $Z_{ab}^{(y)} = \sum_{k=1}^K f_{ab}^{(y)}(k)$. Integrating out the expectation in Eqn.

(12), we obtain

$$g_{ab}(\phi_{ak}) = \frac{f_{ab}^{(y)}(k)}{Z_{ab}^{(y)}\phi_{ak}} - \frac{1}{\phi_a}. \tag{14}$$

Plugging this to Eqn. 3, we obtain the SGRLD update rule for the local parameter ϕ_{ak}

$$\phi_{ak}^* \leftarrow \left| \phi_{ak} + \frac{\epsilon}{2}\left(\alpha - \phi_{ak} + \frac{N}{|\mathcal{V}_n|}\sum_{b\in\mathcal{V}_n} g_{ab}(\phi_{ak})\right) + (\phi_{ak})^{\frac{1}{2}}\xi_{ak}\right|. \tag{15}$$

Here, the $\mathcal{V}_n$ is a random mini-batch of n nodes sampled from $\mathcal{V}^*$. Note that $|\mathcal{V}_n| \ll |\mathcal{V}^*| = N$.

4.3 Scalable local updates for a large number of communities

In some applications, the number of communities can be very large [18] so that the local update becomes very inefficient due to its $\mathcal{O}(K|\mathcal{V}_n|)$ computation per node in $\mathcal{E}_{n_t}$ and also $\mathcal{O}(KN)$ space complexity. In this section, we extend the above algorithm further with a novel approximation in order to make the algorithm scalable in terms of both speed and memory usage even for a very large number of communities which the SVI [1] approach cannot achieve.

Community split: for each node $a \in \mathcal{V}^*$, we first split the community set $\mathcal{K}$ into three mutually exclusive subsets: the *active* set $\mathcal{A}(a)$, the *candidate* set $\mathcal{C}(a)$, and the *bulk* set $\mathcal{B}(a)$ such that $\mathcal{A}(a) \cup \mathcal{C}(a) \cup \mathcal{B}(a) = \mathcal{K}$. Then, sorting the π_a w.r.t. k in descending order, we obtain a new order of communities $k_1, \ldots, k_K$. The active set $\mathcal{A}(a)$ contains communities whose cumulative distribution $F(k_i)$ is less than a threshold $\tau \in (0, 1]$, i.e. $\mathcal{A}(a) = \{k_i \in \mathcal{K} | F(k_i) < \tau\}$. The candidate set $\mathcal{C}(a)$ includes communities which are in the active set of at least one of the neighbors of node a, i.e. $\mathcal{C}(a) = \{k \in \mathcal{K} \setminus \mathcal{A}(a) | \exists b \in \mathcal{N}(a) \text{ s.t. } k \in \mathcal{A}(b)\}$. The bulk set $\mathcal{B}(a)$ contains all the remainder, i.e. $\mathcal{B}(a) = \mathcal{K} \setminus (\mathcal{A}(a) \cup \mathcal{C}(a))$. Here, we use $\mathcal{N}(a)$ to denote the neighbors of node a.

The rationale behind this split scheme is two fold. First, due to *sparsity*, at each node only a small number of communities will have meaningful probability while a large number of communities will have very low probability π_{ak}. We want the communities of low probability to belong to the bulk set, to share a single probability $\pi_{a\boldsymbol{\kappa}}$, and thus to be updated by one-shot for all $k \in \mathcal{B}(a)$. We use $\boldsymbol{\kappa}$ to represent the representative community of a bulk set. Second, due to the *locality*, neighboring nodes are likely to have a similar distribution over communities (after all, the model only assigns high probability to links when the associated nodes have high probability of sampling the same community). That is, when a neighbor of node a has a community k in its active set, this community may be a good candidate to become active for node a as well. By maintaining a candidate set we allow communities to spread efficiently to neighboring nodes and thus through the network.

One-shot update: for communities $k \in \mathcal{B}(a)$, we apply the following approximation of the unnormalized probability in Eqn. (13)

$$\begin{aligned} &f_{ab}^{(y)}(k \in \mathcal{B}(a)) \approx \tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa}) \\ =&\pi_{a\boldsymbol{\kappa}}\left\{\bar{\beta}_a^y(1-\bar{\beta}_a)^{(1-y)}\bar{\pi}_b + \delta^y(1-\delta)^{(1-y)}(1-\bar{\pi}_b)\right\}. \end{aligned} \tag{16}$$

That is, we replace π_{bk} and β_k in Eqn. (13) by $\bar{\pi}_b = \frac{1}{m}\sum_{k\in\mathcal{B}_m(a)}\pi_{bk}$ and $\bar{\beta}_a = \frac{1}{m}\sum_{k\in\mathcal{B}_m(a)}\beta_k$ respectively using a random mini-batch $\mathcal{B}_m(a)$ of size m sampled from $\mathcal{B}(a)$. As a result, all $k \in \mathcal{B}(a)$ share a single value $\tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa})$. Therefore, we can efficiently approximate the normalization constant by

$$\begin{aligned} Z_{ab}^{(y)} =& \sum_{k\in\mathcal{K}} f_{ab}^{(y)}(k) \approx \tilde{Z}_{ab}^{(y)} \\ =&|\mathcal{B}(a)|\tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa}) + \sum_{k\notin\mathcal{B}(a)} f_{ab}^{(y)}(k). \end{aligned} \tag{17}$$

Note that we only sum over $|\mathcal{A}(a) \cup \mathcal{C}(a)|+1$ terms which will be a much smaller size than $|\mathcal{B}(a)|$. Now, to compute the gradient efficiently, we apply the stratified sampling[1] for $\mathcal{V}_n$ by sampling n_1 nodes $\mathcal{V}_1$ from the neighbors $\mathcal{N}(a)$ and n_0 nodes $\mathcal{V}_0$ from non-neighbors $\mathcal{V}^* \setminus \mathcal{N}(a)$ such that $\mathcal{V}_n = \mathcal{V}_1 \cup \mathcal{V}_0$. Then, the sum of gradients for $\mathcal{V}_n$ in Eqn. (15) is obtained by

$$\begin{aligned} &\frac{N}{|\mathcal{V}_n|}\sum_{b\in\mathcal{V}_n} g_{ab}(\phi_{ak}) \\ \approx& c_1 \sum_{b\in\mathcal{V}_1}\left(\frac{\tilde{f}_{ab}^{(1)}(k)}{\tilde{Z}_{ab}^{(1)}\phi_{ak}} - \frac{1}{\phi_a}\right) + c_0 \sum_{b'\in\mathcal{V}_0}\left(\frac{\tilde{f}_{ab'}^{(0)}(k)}{\tilde{Z}_{ab'}^{(0)}\phi_{ak}} - \frac{1}{\phi_a}\right). \end{aligned} \tag{18}$$

Here, we set $c_1 = |\mathcal{N}(a)|/n_1$ and $c_0 = (N - |\mathcal{N}(a)|)/n_0$ to ensure the unbiasedness of the gradient under stratified sampling. Again, it is important to note that all states in $\mathcal{B}(a)$ share a single current value $\phi_{a\boldsymbol{\kappa}}$ and also the same update equation of Eqn. (18). Thus for $\mathcal{B}(a)$ we compute Eqn. (18) only once and update all of them in one-shot. The computation cost becomes $\mathcal{O}(|\mathcal{A}(a) \cup \mathcal{C}(a)||\mathcal{V}_n|)$ per node in $\mathcal{E}_{n_t}$ which we expect to be efficient because $|\mathcal{A}(a) \cup \mathcal{C}(a)| \ll K$ due to sparsity. For $k \notin \mathcal{B}(a)$, we simply replace $\tilde{f}_{ab}^{(y)}(k)$ in Eqn. (18) by $f_{ab}^{(y)}(k)$ in Eqn. (13), and update individually.

[1] Note that, to be more efficient under the approximation, we use a sampling method which is different to the method used in the global update.

Algorithm 1 Pseudo-code for each sampling iteration t

Sample a mini-batch $\mathcal{E}$ of n_t node pairs from $\mathcal{E}^*$
for each node a in $\mathcal{E}$ **do**
  Sample a mini-batch of nodes $\mathcal{V}_n(a) = \mathcal{V}_1(a) \cup \mathcal{V}_0(a)$ from $\mathcal{V}^*$
  Update ϕ_{ak} for all $k \in \mathcal{A}(a) \cup \mathcal{C}(a)$ using Eqn. (18) and Eqn. (15)
  Update $\phi_{a\boldsymbol{\kappa}}$ only for the representative bulk state $\boldsymbol{\kappa}$ using Eqn. (18) and Eqn. (15)
  Sort and normalize to obtain $\{\pi_{ak}\}$ and the cdf $F(k_i)$ for all $|\mathcal{A}(a) \cup \mathcal{C}(a)|+1$ states
  Promote or demote some states using the updated cdf, threshold τ, and neighbor information
end for
for k in a random subset of $\mathcal{K}$ **do**
  Update $\theta_{k\{0,1\}}$ by Eqn. (10) using $\mathcal{E}$ and obtain β_k from $\theta_{k\{0,1\}}$
end for

Promotion and demotion: after updating all $|\mathcal{A}(a) \cup \mathcal{C}(a)|+1$ states (communities), we need to update the community split by promoting (e.g. to active or candidate set) or demoting (e.g. to candidate or bulk set) some of the states. To do this, we sort and normalize $\{\phi_{ak}\}$, and obtain the updated cdf $F(k_i)$. Then, we update $\mathcal{A}(a)$, $\mathcal{C}(a)$, and $\mathcal{B}(a)$ based on the threshold τ and based on the communities that are active in the neighboring nodes. In particular, if the cdf of the bulk state $F(\boldsymbol{\kappa})$ is less than the threshold, we promote some states in the bulk set by a random sampling. In this case, the number of states to promote is equal to $\text{int}((\tau - F(\boldsymbol{\kappa}_{-1}))/\pi_{a\boldsymbol{\kappa}})$. Here $\boldsymbol{\kappa}_{-1}$ denotes a state just left to the $\boldsymbol{\kappa}$ in the sorted community sequence. We sample a state from the pool of states that are yet not represented anywhere in the graph. The reason is that we wish to avoid creating disconnected communities of nodes, which we believe represent suboptimal local modes in the posterior distribution. Finally, we check which states in $\mathcal{B}(a)$ can be promoted to $\mathcal{C}(a)$ by checking the neighboring nodes. We provide the pseudo code of the above algorithm in the Algorithm 1.

5 Experiments

We evaluate the efficiency and accuracy of our algorithm on five datasets [18]: Synthetic [1], US-AIR, NETSCIENCE, RELATIVITY, and HEP-PH, summarized in Table 1. (The last column is the percentage of link edges among all possible edges.) We compare four algorithms. As exact batch-mode MCMC methods, we use collapsed Gibbs sampling (CGS) and Langevin Monte Carlo (LMC). We also compare to SVI [1] as a state-of-the-art method in variational Bayes. Finally, two of our algorithms are tested, one with and the other without the approximation for large communities. We call these SGMC and SGMC-M, respectively.

Table 1: Datasets

Name	# of nodes	%
Synthetic	75	30
US-AIR	1.1k	1.2
NETSCIENCE	1.6K	0.3
RELATIVITY	5.2K	0.05
HEP˙PH	12k	0.16

We used $\alpha = 1/K$ and $\eta = 1$ for all of the models and for all experiments unless otherwise stated. For the stepsize annealing schedule we used $\varepsilon_t = (\tau_0 + t)^{-\kappa}$ with $\kappa = 0.5$ and $\tau_0 = 1024$ [2]. For the stratified sampling of the global update in SVI and SGMCs, we used m such that the size of non-link edges N/m to be $30 < N/m < 100$. And for the mini-batch size of the stratified sampling of the local update in SGMCs, we used 20 samples with 10 from neighbors and 10 from non-neighboring nodes. For SGMC-M, we used the threshold $\tau = 0.9$ by default unless otherwise stated. Also, for the held-out test set, we used 1% of the total links and non-links.

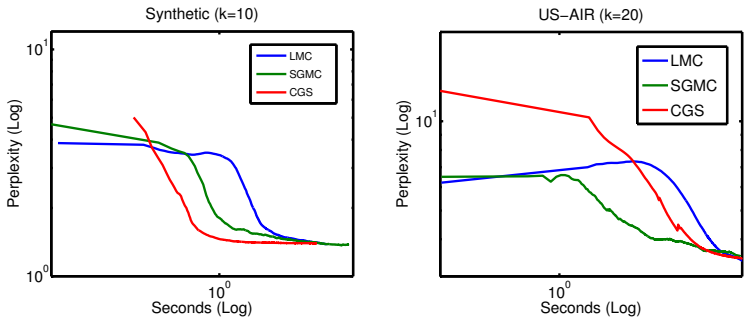
As the performance metric, we use perplexity which is defined as exponential of the negative average log-likelihood of the data. Given a collection of T samples of the model parameters $\{\beta_t\}$ and $\{\pi_t\}$, the averaged perplexity on the held-out test set $\mathcal{E}_h$ is

$$\text{perp}_{\text{avg}}(\mathcal{E}_h|\{\beta_t\},\{\pi_t\}) = \exp\left(-\frac{\sum_{(a,b)\in\mathcal{E}_h}\log\{(1/T)\sum_{t=1}^{T}p(y_{ab}|\beta_t,\pi_t)\}}{|\mathcal{E}_h|}\right) \tag{19}$$

5.1 Results

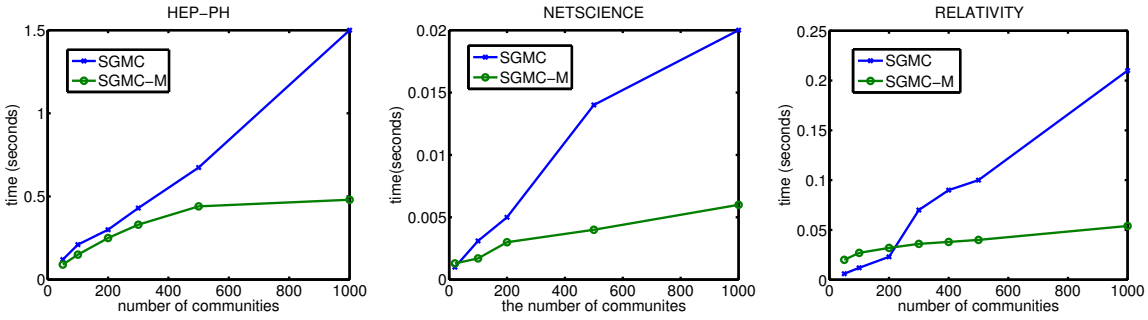
Comparison to exact batch MCMC: We first show the accuracy of our algorithm in comparison to exact batch-mode MCMC algorithms (CGS and LMC). For this, we use two relatively small datasets, Synthetic and US-AIR, due to the slow speed of the batch algorithms. The results are shown in Fig. 1a and Fig. 1b.

As expected, for the smaller dataset (Synthetic) in Fig. 1a, we see that CGS converges very fast. However, it is interesting to observe that our stochastic gradient sampler (SGMC) using fixed step-size converges to the same level of accuracy in comparable time, whereas LMC converges much slower than both the collapsed



(a) (b)

Figure 1: Convergence of perplexity on (a) Synthetic and (b) US-AIR datasets



(a) (b) (c)

Figure 2: (a) Wall-clock time per iteration over increasing community sizes, on (a) HEP-PH, (b) NETSCIENCE and (c) RELATIVITY datasets

Gibbs sampler and our algorithm due to its full gradient computation and the Metropolis-Hastings accept-reject step. As we move to a larger network (US-AIR) in Fig. 1b, we begin to see that our stochastic gradient sampler outperforms in speed the collapsed Gibbs sampler as well as the Langevin Monte Carlo. It is interesting to see that the approximation error of our algorithm due to the finite step size and the absence of accept-reject tests is negligible compared to the perplexity of the exact MCMC.

Effect of our approximation for large communities: In Fig. 2a and Fig. 2b, Fig.2c on three large datasets, HEP-PH, NETSCIENCE and RELATIVITY, we show the speed-up effect of our approximate method (SGMC-M) compared to the SGMC without the approximation. Here we measure the time per iteration for various community size $K = [30, 50, 100, 200, 300, 500, 1000]$ and set the threshold to $\tau = 0.9$. As shown, we can see that the approximate method SGMC-M only slightly increases the wall-clock time per iteration even if the community size increases. However, without the approximation (SGMC), the time per iteration increases linearly w.r.t. the community size. In fact, we can obtain more time savings as the community size increases further because the level of sparsity, i.e. the number of communities for which each node has non-negligible probability of participation, does not change much when we increase K.

Furthermore, it is interesting to see in Fig. 3a, Fig. 3b and Fig. 3c that we do not lose much accuracy despite the approximation. In particular, for Fig. 3b, the SGMC-M performs as good as the SGMC. Although for Fig. 3c the SGMC-M performs worse than the SGMC, it still outperforms the SVI. Note that the results are based on converged perplexity which SGMC-M will reach much faster. The figures also reveal some interesting facts. First, the predictive accuracy is dominated by SGMC for all choices of K. Second, the curve of SVI has a V-shape indicating that the optimal value for K is in between the minimum and maximum value of K we tested. However, for SGMC the accuracy remains relatively stable as we increase K, making it less sensitive to the choice of this hyperparameter.

In Fig. 4, we show the convergence of perplexity over wall-clock time on several datasets using SGMC, SGMC-M and SVI. Note that we do not include the SVI method in Fig. c and Fig. 4d where we use a large community size, since the perplexity tends to

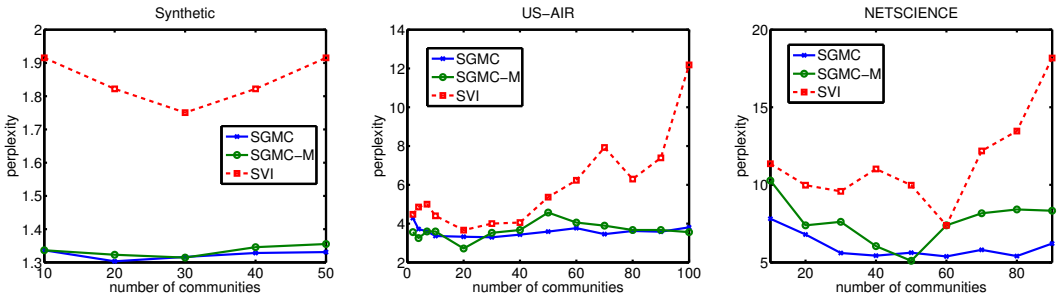


Figure 3: Converged perplexity for various community sizes on (a) Synthetic , (b) US-AIR and (c) NETSCIENCE datasets

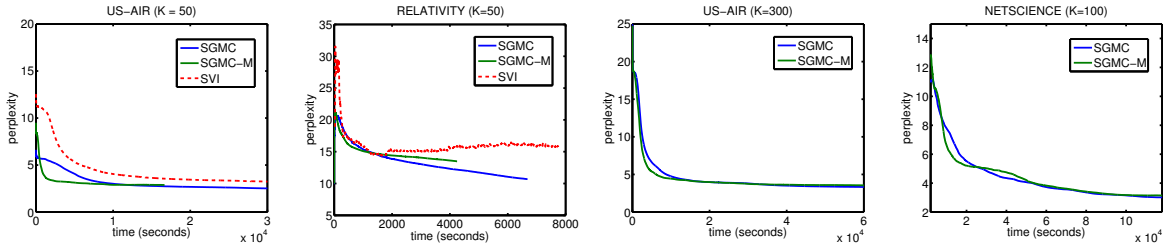


Figure 4: The change of perplexity over time on (a) US-AIR (K=50), (b) RELATIVITY (K=50), (c) US-AIR (K=300) and (d) NETSCIENCE (K=100) datasets. For (a) and (b), we compare the performance among three methods: SGMC, SGMC-M and SVI. However, for (c) and (d) we don't include the results for SVI because the perplexity for SVI is hard to compare to our methods.

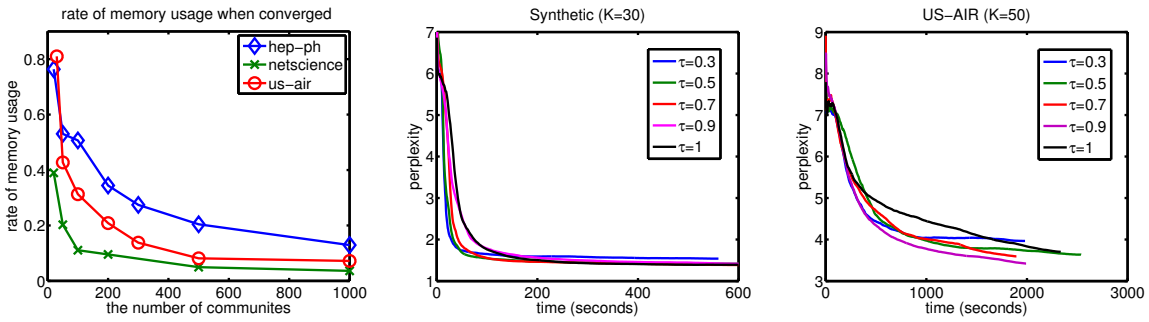
become very large and thus makes it incomparable to our methods. In general, we have two main observations. First, the approximate method SGMC-M dominates other methods during early stages, but eventually SGMC reaches lower perplexity than SGMC-M. Second, in Fig. 4b and Fig. 4b, both SGMC and SGMC-M reach much lower perplexity than SVI.

In Fig. 5a, we show the efficiency of SGMC-M in terms of memory usage. In this experiment, we set the threshold $\tau = 0.9$ for all datasets. As shown, the memory usage (i.e. $|\mathcal{A}(a) \cup \mathcal{C}(a)|/K$) of SGMC-M decreases as the number of communities increases. This is because the sparsity does not change even if we increase the community size K. Note that the memory usages of SVI and SGMC are always 100%. This is a significant feature for large scale networks because without the approximation the space complexity is $\mathcal{O}(KN)$ where both K and N can be very large [18]. It is also interesting to see that at every node 90% of the total density is allocated to only about $10\% \sim 20\%$ of the communities (e.g., for $K = 500, 1000$).

Lastly, we investigate the effect of the threshold τ and the results are shown in Fig. 5b and Fig. 5c using Synthetic and US-AIR datasets. As shown, with $\tau = 0.9$ and $\tau = 1$, we obtain the best result. It is interesting because the memory usage of SGMC-M is only a half of SGMC without the approximation ($\tau = 1$). As expected, as we decrease the threshold, smaller communities are represented in the active and candidate set and thus we lose some accuracy while gaining some speed-up.

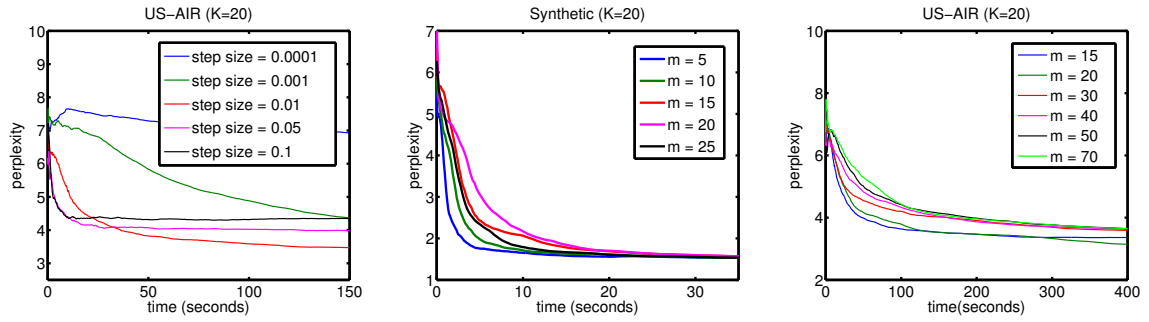
Effect of step sizes: SG-MCMC converges in theory as the step size goes to zero. Although we have used decreasing step sizes in the above experiments, it will be interesting to see how fixed step sizes affect the algorithm, because in practice we cannot decrease the step size to zero. One of the result is shown in Fig. 6a. The figure clearly reveals the trade-off between a large step size (leading to a large bias) and a small step size (leading to slow mixing). For the US-AIR dataset a step size 0.01 seems to work best.

Mini-batch size and computational efficiency: Depending on the number of edges/nodes sampled in a mini-batch, there is a clear trade-off between the computational cost per update and the variance of an update (i.e. for stochastic gradients). Our SG-MCMC includes two sampling steps for each iteration, one for edges and the other for nodes (line 1 and 3 in Algorithm 1). Since we are using a "stratified random



(a) (b) (c)

Figure 5: (a) Memory usage over different community sizes and datasets. The memory usage is defined as the ratio $(|\mathcal{A}|+|\mathcal{C}|+1)/K$. (b) and (c) Convergence of perplexity for various threshold values τ on Synthetic and US-AIR datasets.



(a) (b) (c)

Figure 6: (a) Effects of step size. (b) and (c) Effects of mini-batch size. We fix the step size and mini-batch size during the entire training.

node sampling" strategy, it is important to choose a proper m in order to obtain both fast convergence and low variance. We tested the effects of the parameter m. The results are shown in Fig. 6b and Fig. 6c. As we can see, setting m to 5 achieves the best performance for the synthetic dataset while $m = 20$ performs best for the US-AIR data set.[2] In practice, setting m in such a way that the corresponding mini-batch size becomes between 30 and 100 results in good performance, i.e. if we have 10K nodes in total, we could set m to a value between 100-350.[3]. Overall, SG-MCMC takes $O(n^2K)$ computational time[4] for each iteration, where $n = N/m$ and N is the total number of the nodes.

[2]Note that setting m equals to 5 for the synthetic data set is roughly equivalent to setting the mini-batch size to $75/m = 15$.

[3]For the experiments with SVB we choose $m = 100$ to make the comparison fair.

[4]This complexity is computed by assuming that (i) we use the "stratified node sampling" and (ii) the average number of links per node is bounded below by $n = N/m$.

6 Conclusion and Future Work

In this paper we have developed a new scalable MCMC algorithm based on stochastic gradient computations for assortive mixed membership stochatic blockmodels (a-MMSB). The algorithm represents a natural extension of stochastic gradient Riemannian Langevin dynamics (SGRLD) [2] to a-MMSBs. In line with the results reported in [2] for LDA, SGRLD also significantly outperforms its stochastic variational Bayesian counterpart. As was shown in [18], SGRLD algorithms are particularly suited for distributed implementation. We are currently working towards a distributed implementation of our algorithm on a HPC infrastructure allowing us to perform full Bayesian inference on the very large "Friendster" network with almost two billion edges. Initial results show that we achieve good perplexity and sample to convergence on the full network.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1216045, as well Google and Facebooks gifts.

References

[1] P. Gopalan, D. Mimno, S. Gerrish, M. Freedman, and D. Blei. Scalable Inference of Overlapping Communities. *Advances in Neural Information Processing Systems*, 2012.

[2] S. Patterson and Y. W. Teh. Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex. *Advances in Neural Information Processing Systems*, 2013.

[3] M. Welling and Y. W. Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. *International Conference on Machine Learning*, 2011.

[4] D. Blei, A. Ng, M. Jordan, and J. Lafferty. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 2003.

[5] S. Ahn, B. Shahbaba, and M. Welling. Distributed Stochastic Gradient MCMC. *International Conference on Machine Learning*, 2014.

[6] S. Ahn, A. Korattikara, and M. Welling. Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring. *International Conference on Machine Learning*, 2012.

[7] M. Hoffman, D. Blei, C. Wang, and J. Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 2013.

[8] E. Airoldi, D. Blei, S. Fienberg, and E. Xing. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research.*, 2008.

[9] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov*, 2007.

[10] RITA. U.S. Air Carrier Trafic Statistics, Bur. Trans. Stats, 2010.

[11] T. Griffiths and M. Steyvers. Finding Scientific Topics. Proceedings of the National academy of Sciences, 2004.

[12] M. Hoffman, D. Blei, and F. Bach, Online Learning for Latent Dirichlet Allocation. *Advance in Neural Information Processing systems*, 2010.

[13] M. Wainwright and M. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 2008.

[14] R. Neal. Probabilistic inference using Markov chain Monte Carlo methods. *Technical report CRG-TR 93-1 Department of Computer Science, Universit of Toronto*, 1993.

[15] M. Girolami, and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2011.

[16] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. Machine learning, 1999.

[17] R. M. Neal. MCMC using Hamiltonian dynamics. In Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (eds.), *Handbook of Markov Chain Monte Carlo*. Chapman & Hall /CRC Press, 2010

[18] Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/

[19] S. Ahn, A. Korattikara, N. Liu, S. Rajan, and M. Welling, Large-Scale Distributed Bayesian Matrix Factorization using Stochastic Gradient MCMC, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2015.