JMLR: Workshop and Conference Proceedings 13: 63-78
2nd Asian Conference on Machine Learning (ACML2010), Tokyo, Japan, Nov. 8–10, 2010.

Efficient Collapsed Gibbs Sampling For Latent Dirichlet Allocation

Han Xiao XIAOH@IN.TUM.DE
Thomas Stibor STIBOR@IN.TUM.DE
Department of Informatics
Technical University of Munich, GERMANY

Editor: Masashi Sugiyama and Qiang Yang

Abstract

Collapsed Gibbs sampling is a frequently applied method to approximate intractable integrals in probabilistic generative models such as latent Dirichlet allocation. This sampling method has however the crucial drawback of high computational complexity, which makes it limited applicable on large data sets. We propose a novel *dynamic sampling* strategy to significantly improve the efficiency of collapsed Gibbs sampling. The strategy is explored in terms of efficiency, convergence and perplexity. Besides, we present a straight-forward parallelization to further improve the efficiency. Finally, we underpin our proposed improvements with a comparative study on different scale data sets.

Keywords: Gibbs sampling, Optimization, Latent Dirichlet Allocation

1. Introduction

Latent Dirichlet allocation (LDA) is a generative probabilistic model that was first proposed by Blei et al. (2003) to discover topics in text documents. LDA is based on the assumption that a document is a mixture of different topics and can be considered as a hierarchical generative process of creating documents given the topics. By reversing the generative process of LDA, one obtains a predictive model by means of the posterior distribution. However, this "reversal" requires an estimation of an intractable integral and hence approximation algorithms are required. There are three estimation algorithms for LDA. Blei et al. (2003) proposed a variational Expectation-Maximization algorithm, which iteratively executes the E-step and M-step. The E-step estimates the topic distribution of each training document using current model parameters, and the M-step updates the model parameters. This approach is prone to local optima due to the wiggly likelihood function. Minka and Lafferty (2002) showed that variational EM can lead to inaccurate inferences and biased learning, they developed the Expectation-Propagation algorithm that leads to higher accuracy at comparable cost. Griffiths and Steyvers (2004) proposed the Collapsed Gibbs Sampling (CGS), which is a Markov-chain Monte Carlo method. Due to the fact that CGS is a straight-forward approach and rapidly converges to a known ground-truth, it has been widely used in many LDA variants. However, CGS has the crucial drawback of high computational complexity, which makes it inefficient on large data sets.

In this paper, we propose an **E**fficient **C**ollapsed **G**ibbs **S**ampling algorithm (ECGS) to speedup the estimating procedure of LDA. In particular, we introduce a *dynamic sampling* strategy to reduce the total number of sampling times in each iteration. By exploiting the fact that the posterior distribution is usually sparse, the dynamic sampling can automatically adapt the sampling times for each word to avoid unnecessary sampling, while converging to the same ground truth solution as the standard CGS. Experiments on real-world data sets suggest that ECGS is 3-6 times faster than standard CGS and 2-4 times faster than previously published fast sampling methods. Furthermore, we parallelize ECGS using OpenMP, which gives a superior efficiency on shared memory system. For the sake of clarity the paper is structured as follows: In Sect. 2, previous works on optimization and parallelization of CGS are briefly reviewed. Sect. 3 describes the standard CGS algorithm and points out the optimization part. In Sect. 4 the proposed ECGS algorithm is explained and different sampling strategies are presented. Experiments on real-world data sets are presented in Sect.5. Sect. 6 concludes.

2. Related Work

In prior works different implementations and improvements have been explored to overcome the limitation of scalability of CGS. The earliest C++ implementation is GibbsLDA[1], and is therefore widely used as a baseline in comparative studies. Porteous et al. (2008) implemented FastLDA[2] in C, which draws equivalent samples as GibbsLDA but requires on average significantly less operations per word. They exploited the fact that the posterior distribution $P(z|w)$ is sparse for most of the words w and topics z. As a consequence, they constructed an adaptive upper bound of the normalization factor of $P(z|w)$, and hence obtained an efficient multinomial sampler. Moreover, FastLDA produces equivalent results as the standard CGS. Yao et al. (2009) presented SparseLDA[3] to further improve the complexity of Gibbs sampling. They divided the full conditional probability mass into three parts and employed an approximate sampling scheme to change the document-topic count and word-topic count respectively. Although the relative speedup compared to standard CGS seems promising, SparseLDA is implemented in JAVA, which makes it much slower than other C/C++ implementations, thus it is hard to gauge the absolute performance of SparseLDA. Furthermore, Canini et al. (2009) proposed two online inference algorithms: incremental Gibbs sampler and particle filter. In incremental Gibbs sampler, only particular words in the "rejuvenation sequence" are sampled in each iteration. Thus, the choice of rejuvenation steps determines the runtime of the incremental Gibbs sampler. The particle filter introduced a resampling strategy to optimize the Gibbs sampler, however, to implement this algorithm efficiently a special data structure has to be designed and maintained in the memory.

Even after exploring a great deal of optimization approaches, training LDA using CGS still remains computationally expensive, therefore an effective parallelization becomes a natural choice. Newman et al. (2007) presented two synchronous methods, AD-LDA and HDLDA, to perform distributed CGS. Wang et al. (2009) implemented AD-LDA by using MPI and MapReduce and called the new approach PLDA. They observed that their PLDA[4]

1. `http://gibbslda.sourceforge.net/`
2. `http://www.ics.uci.edu/~iporteou/fastlda/`
3. `http://mallet.cs.umass.edu`
4. `http://code.google.com/p/plda/`

implementation scales well on up to 64 distributed machines. Recently, Yan et al. (2009) proposed parallel LDA on Graphics Processing Units (GPU). To address limited memory constraints on GPUs, their LDA-GPU introduced a novel data partitioning scheme that effectively reduces the memory cost. LDA-GPU can be regarded as an extension of AD-LDA by using the data partition in local sampling and inserting synchronization steps within an iteration.

In this paper, we take advantage of the sparse posterior distribution $P(z|w)$ and propose the ECGS algorithm which is based on two novel optimization strategies. On real-world data sets, the ECGS-*Dynamic* provides 3-6 times speedup against the standard CGS and converges to the same ground truth distribution given by the latter. ECGS-*Shortcut* further improve the speedup to 5-9 times, while provides a suboptimal solution. Moreover, our parallel ECGS-OpenMP enjoys approximately linear speedup on shared memory system.

3. Standard CGS

Before introducing the ECGS algorithm, we briefly review the standard CGS algorithm. The notations used in this paper are summarized in Table 1. Besides, the following name convention is used throughout this paper. The occurrences of a word is termed *token*, the unique words are termed *types.* For example,“The dog barks at the cat” has 5 tokens but 4 types. There are two tokens of the type “the”.

Symbol	Description
K	number of topics
D	number of documents
W	number of tokens
V	number of types
N_d	number of types in document d
N_{di}	number of the ith type in document d
w_{di}	the ith type in document d
z_{dij}	the topic associated with the jth token of w_{di}
C_{vk}	number of type v which are assigned with topic k
C_{dk}	number of topics k in document d
$\mathcal{I}_{di}$	sampling times of w_{di} in one Gibbs Sampling iteration
$\mathcal{S}_{di}^t$	sampling rate of type w_{di} in iteration t
$\bar{\mathcal{S}}^t$	average sampling rate in iteration t
Γ_{di}	parameter vector of the multinomial distribution $P(\mathcal{I}_{di}\|\Gamma_{di})$, it has N_{di} entries: $[\Gamma_{di1}, \cdots, \Gamma_{diN_{di}}]$
α, β	Dirichlet priors
γ	dumping factor

Table 1: Notations used in ECGS algorithm. A symbol that is in bold refers to an array. Arrays can be vectors or matrices.

The standard CGS iteratively samples a topic for every token in the training set from a full conditional probability. There are three steps involved in this procedure. First, the unnormalized posterior distribution is computed as follows:

$$P(z_{di} = k | w_{di} = v, \mathbf{W}_{\neg w_{di}}, \mathbf{Z}_{\neg z_{di}}, \alpha, \beta) \propto (C_{dk} + \alpha) \frac{C_{vk} + \beta}{\sum_{v'} C_{v'k} + V\beta}. \tag{1}$$

Second, a random variable x is sampled from a uniform distribution, namely $x \sim \texttt{uniform}(0, \sum_{z=1}^{K} P(z|w))$. The final step is to find the interval that x falls into. That is, finding $\hat{k}$ such that $\sum_{z=1}^{\hat{k}-1} P(z|w) < x < \sum_{z=1}^{\hat{k}} P(z|w)$, where $\hat{k}$ is the sampled topic of the current token. The time complexity for each iteration is $\mathcal{O}(KW)$ [5]. The pseudo-code of the standard CGS algorithm[6] is provided in Figure 1, which is also the basement of our proposed two novel algorithms as we later shall see. By studying the code of standard CGS, one can observe that there are typically four nested for-loops involved. Porteous et al. (2008) and Yao et al. (2009) focused on optimizing the computation in the 4*th* loop (line 7 to 8) by eliminating topics having small probabilities. The parallel CGS methods proposed by Yan et al. (2009) and Wang et al. (2009) can be seen as optimizing the 1*st* and 2*nd* loop, as they divide the data set on different processors and reduce the total number of D and N_d on each processor. Alternatively, our ECGS algorithm mainly optimizes the 3*rd* loop (line 4 to 12) by reducing the number of iterations N_{di}.

4. ECGS Algorithm

In this section, two optimization strategies that form the ECGS algorithm are presented. The underling philosophy of the two strategies is the same: reducing the number of iterations N_{di} in the 3*rd* loop. For the sake of clarity, we first introduce a relative simple strategy: shortcut sampling, which is helpful to understand the more elaborate one: dynamic sampling.

4.1 ECGS-*Shortcut*

To introduce our optimization strategies, we first define the *sampling rate* in CGS iteration t for type w_{di} as follows:

$$\mathcal{S}_{di}^{t} = \frac{\mathcal{I}_{di}}{N_{di}}, \text{ where } \mathcal{S}_{di}^{t} \in [0,1]. \tag{2}$$

$\mathcal{I}_{di}$ is defined as the sampling times in 3*rd* loop for type w_{di}. Additionally, the *average sampling rate* for a training set is defined as:

$$\bar{\mathcal{S}}^{t} = \frac{\sum_{d=1}^{D} \sum_{i=1}^{N_d} \mathcal{S}_{di}^{t} N_{di}}{W}, \text{ where } \bar{\mathcal{S}}^{t} \in [0,1]. \tag{3}$$

5. For the sake of clarity, we define the complexity of invoking the multinomial random number generator `multinomial()` as $\mathcal{O}(1)$. This ignores the internal computational cost of the generator.
6. The `foreach` statement randomly visits each element in the set to ensure that the algorithm samples from correct posterior. Moreover, in many LDA algorithms which employ CGS, inner for-loops in Figure 1 (line 2 and 4) are merged by cycling over tokens in the order they occur. Such an implementation is nevertheless equivalent to ours, yet in a different programming style.

```
1  foreach $d \in \{1, \cdots, D\}$
2      foreach $i \in \{1, \cdots, N_d\}$
3          $v \leftarrow w_{di}$, $\mathcal{I}_{di} \leftarrow N_{di}$
4          foreach $j \in \{1, \cdots, \mathcal{I}_{di}\}$
5              $\hat{k} \leftarrow z_{dij}$
6              $C_{d\hat{k}} \leftarrow C_{d\hat{k}} - 1$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} - 1$
7              for $k = 1$ to $K$ do
8                  $\rho_k \leftarrow \rho_{k-1} + (C_{dk} + \alpha) \times (C_{kv} + \beta) / (\sum_{v'} C_{v'k} + V\beta)$
9              $x \sim$ Uniform$(0, \rho_K)$
10             $\hat{k} \leftarrow$ BinarySearch$(\hat{k} : \rho_{\hat{k}-1} < x < \rho_{\hat{k}})$
11             $C_{d\hat{k}} \leftarrow C_{d\hat{k}} + 1$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} + 1$
12             $z_{dij} \leftarrow \hat{k}$
```

Figure 1: Pseudo-code of standard CGS algorithm of one iterated step which is denoted as one *CGS iteration* in the rest of the paper. Note, that $\{\rho_1, \rho_2, \cdots, \rho_K\}$ is an increasing sequence, we thus employ binary search to find the suitable $\hat{k}$. This simple trick improves the efficiency of CGS, however it is surprisingly ignored in previous implementations (a simple linear list search is employed in the other implementations). Since we do not focus on optimizing the algorithm of generating multinomial random number, we hereinafter use $\hat{k} \sim$ `multinomial`$(\cdot)$ to denote the code from line 7 to 10.

Intuitively, $\bar{\mathcal{S}}^t$ can be explained as the sampling coverage of the training set in CGS iteration t. The larger $\bar{\mathcal{S}}^t$ is, the more tokens are covered in the sampling procedure, and thus the slower the running speed of the algorithm. Obviously, $\mathcal{S}^t_{di}$ and $\bar{\mathcal{S}}^t$ in standard CGS have the constant value 1 regardless of the number of iterations and the type, as it samples topics for all tokens in the training set. It is also clear that $\bar{\mathcal{S}}^t = 1$ is the highest sampling rate and gives the slowest running speed. In general, we want a small value of $\bar{\mathcal{S}}$ so that the algorithm runs faster. This can be achieved by sampling only a topic for every type in each document, rather than sampling a topic for every token in each document. The sampled topic is then assigned to all tokens of that type in the document. Thus, all repetitive tokens in one document have the same topic. This strategy (termed ECGS-*Shortcut*) can significantly reduce the sampling rate, as it introduces a shortcut sampling that only iterates over the types in each document. The pseudo-code of ECGS-*Shortcut* is given in Figure 2.

The benefit of ECGS-*Shortcut* is obvious. Due to the large amount of repetitive tokens in the real-world data set, it reduces $\mathcal{S}^t_{di}$ from 1 to $1/N_{di}$, and $\bar{\mathcal{S}}^t$ from 1 to $\sum_d N_d/W$. As a consequence, it provides a significant speedup. The time complexity for each CGS iteration is $\mathcal{O}(K \sum_d N_d)$. Note, that the actual speedup depends on the redundancy of the data set. Nevertheless, ECGS-*Shortcut* may only give a suboptimal solution, as the cursory sampling may miss the optimum solution. To address this problem, we introduce the dynamic sampling strategy, meanwhile keep a high running speed.

```
1 foreach $d \in \{1, \cdots, D\}$
2     foreach $i \in \{1, \cdots, N_d\}$
3         $v \leftarrow w_{di}$, $\hat{k} \leftarrow z_{di1}$
4         $C_{d\hat{k}} \leftarrow C_{d\hat{k}} - N_{di}$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} - N_{di}$
5         $\hat{k} \sim$ Multinomial($P(z|w_{di})$)
6         $C_{d\hat{k}} \leftarrow C_{d\hat{k}} + N_{di}$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} + N_{di}$
7         $z_{di1} \leftarrow \hat{k}$
```

Figure 2: Pseudo-code of ECGS-*Shortcut* of one iterated step. The 3*rd* for-loop is eliminated by setting $\mathcal{I}_{di} = 1, \mathcal{S}^t_{di} = 1/N_{di}$ for type w_{di}.

4.2 ECGS-*Dynamic*

The intuition behind ECGS-*Dynamic* is to automatically adapt $\bar{\mathcal{S}}^t$ over the iterations, where $\bar{\mathcal{S}}^t \in [\sum_d N_d/W, 1]$. Loosely speaking, we start CGS with a high sampling rate (e.g. $\bar{\mathcal{S}}^1 = 1$), such that every token is involved in the sampling procedure. This ensures that the solution space is sufficiently explored at the beginning. Then, we gradually reduce $\bar{\mathcal{S}}^t$ over the iterations, that is, $\bar{\mathcal{S}}^1 \geqslant \bar{\mathcal{S}}^2 \geqslant \cdots \geqslant \bar{\mathcal{S}}^t$.

Rather than directly decrementing $\bar{\mathcal{S}}^t$ by a predefined function, our algorithm is distinctive in two aspects. First, for each type in the data set, we specify the "sampling times" $\mathcal{I}_{di}$ of the type as a random variable. Second, we estimate the probability distribution of $\mathcal{I}_{di}$ in the Gibbs sampling procedure. Specifically, we model $P(\mathcal{I}_{di})$ as a multinomial distribution with a parameter vector Γ_{di}, where Γ_{di} has N_{di} entries. Instead of iterating a constant number of times to sample topics for w_{di}, the algorithm first draw the "sampling times" from a multinomial distribution, namely $\mathcal{I}_{di} \sim P(\mathcal{I}_{di}|\Gamma_{di})$, and then iterate $\mathcal{I}_{di}$ times to sample topics for w_{di}. As $\mathcal{I}_{di} \in \{1, \cdots, N_{di}\}$, the sampling rate $\mathcal{S}^t_{di}$ can be either low (when $\mathcal{I}_{di}$ close to 1) or high (when $\mathcal{I}_{di}$ close to N_{di}). After sampling $\mathcal{I}_{di}$ times, the parameter vector Γ_{di} is updated according to the number of unique topics drawn. This sampling strategy is therefore named as dynamic sampling, as the value of $\mathcal{I}_{di}$ is dynamically adapted over the iterations. The pseudo-code of ECGS-*Dynamic* is given in Figure 3.

An illustrated comparison of different sampling procedures is provided in Figure 4. Given for instance an 8-tokens document with 5 times "cat" and 3 times "dog". The standard CGS samples 8 times for every token. ECGS-*Shortcut* samples 2 times only, since there are only two types in the document. In ECGS-*Dynamic*, Γ_{cat} is initialized as $[0, 0, 0, 0, 1]$, the sampling procedure for each type involves two steps. For instance, to sample the topics for type "cat", we first draw a "sampling times" from `multinomial`$(0, 0, 0, 0, 1)$ (Figure 3 line 4), in this case the only possible outcome is $\mathcal{I}_{\text{cat}} = 5$. Thus, we iteratively sample the topics of "cat" for 5 times, which covers all five tokens in this document, and $\mathcal{S}^1_{\text{cat}} = 1$. Next, we find this 5-times sampling results in 3 unique topics, that is, $|\mathcal{M}| = 3$. Consequently, Γ_{cat} is updated to $[0, 0, 1, 0, 1]$, the new Γ_{cat} will be used for generating the "sampling times" in the 2*nd* CGS iteration. If $\mathcal{I}_{\text{cat}} = 3$ in the 2*nd* CGS iteration, then we save the time cost by excluding two tokens of "cat" from the sampling procedure. Ergo, we get a sampling rate for the 2*nd* CGS iteration of $\mathcal{S}^2_{\text{cat}} = 0.6$. On the other hand, if $\mathcal{I}_{\text{cat}} = 5$ is drawn again,

```
foreach $d \in \{1, \cdots, D\}$
    foreach $i \in \{1, \cdots, N_d\}$
        $v \leftarrow w_{di}$, $\mathcal{M} \leftarrow \emptyset$
        $\mathcal{I}_{di} \sim$ multinomial($P(\mathcal{I}_{di}|v, \Gamma_{di})$)
        foreach $j \in \{1, \cdots, \mathcal{I}_{di}\}$
            $\hat{k} \leftarrow z_{dij}$
            $C_{d\hat{k}} \leftarrow C_{d\hat{k}} - 1$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} - 1$
            $\hat{k} \sim$ multinomial($P(z|w_{di})$)
            $C_{d\hat{k}} \leftarrow C_{d\hat{k}} + 1$, $C_{v\hat{k}} \leftarrow C_{v\hat{k}} + 1$
            $z_{dij} \leftarrow \hat{k}$
            $\mathcal{M} \leftarrow \mathcal{M} \cup \{\hat{k}\}$
        $u \leftarrow |\mathcal{M}|$, $\Gamma_{diu} \leftarrow \Gamma_{diu} + 1$
```

Figure 3: Pseudo-code of ECGS-*Dynamic* by modeling $\mathcal{I}_{di}$ as a random number. After the $3rd$ loop (line 5 to 11) is finished, we add 1 to the uth entry of Γ_{di} (line 12), where u is given by $|\mathcal{M}|$, the number of unique topics sampled in the $3rd$ loop, and $|\mathcal{M}| \in \{1, \cdots, \min\{\mathcal{I}_{di}, K\}\}$.

then we have to sample 5 times for the "cat", which will give $\mathcal{S}^2_{\text{cat}} = 1$. This procedure continues and consequently more and more tokens are excluded. To sum up, the sampling times $\mathcal{I}_{di}$ for each type is generated from a multinomial distribution, which is the major difference between ECGS-*Dynamic*, ECGS-*Shortcut* and standard CGS algorithms.

To achieve a high sampling rate at the beginning, Γ_{di} is initialized as $[0, \cdots, 0, \gamma]$ for every type, where γ is the damping factor and $\gamma \in \mathbb{N}^+$. That is, the $N_{di}th$ entry has a positive value and all other entries are 0. This setting ensures a high sampling rate (close to 1) and an exhaustive exploration in solution space at the beginning. The damping factor γ controls the descending speed of the sampling rate. A large value of γ prevents the sampling rate decreasing too fast and preserves a high sampling rate over several iterations, whereas a small value of γ reduces the sampling rate rapidly to a low value after a few iterations. Additionally, we can build standard CGS and ECGS-*Shortcut* from ECGS-*Dynamic*, by simply initializing Γ_{di} for every word to $[0, \cdots, 0, \infty]$ and $[1, 0, \cdots, 0]$ respectively.

By observing Figure 3 and the example above, one may notice that $|\mathcal{M}|$ plays a key role in the dynamic sampling, as it controls the updated parameters of Γ. Indeed, $|\mathcal{M}|$ is affected by the sparseness of $P(z|w)$ in a subtle way. To study the relation between sparseness of $P(z|w)$ and $|\mathcal{M}|$, we derive the expectation of $|\mathcal{M}|$ as:

$$E(|\mathcal{M}|) = \sum_{k=1}^{K} \Big(1 - \big(1 - P(z = k|w)^{\mathcal{I}_{di}}\big)\Big). \tag{4}$$

The proof of (4) is given in Appendix. Intuitively, $E(|\mathcal{M}|)$ indicates the entry of Γ_{di} which is updated. Therefore, it also affects the value of $\mathcal{I}_{di}$ which is used in the next iteration. Furthermore, we use the entropy of $\hat{k}$ to describe the sparseness of $P(z|w)$, which is given

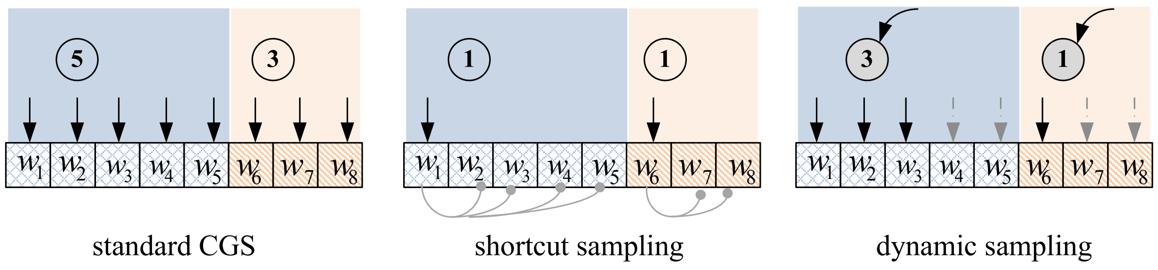


Figure 4: Sampling procedure of standard CGS, ECGS-*Shortcut* and ECGS-*Dynamic*. Each box represents a token, the box with same color has the same type, different colors/patterns indicates different types. The number in the circle is the sampling times for a certain type. Each arrow denotes invoking the function $x \sim$`multinomial()` for one times. In this example, the document contains 8 tokens and 2 types. Standard CGS samples 8 times (for every token) per iteration, ECGS-*Shortcut* samples 2 times (only for 2 types) per iteration, and then assigns the same topic to all tokens that have same type. By contrast, the sampling times in ECGS-*Dynamic* is a random variable, which is controlled by the multinomial distribution $P(\mathcal{I}_{di}|\Gamma_{di})$.

by:

$$H(\hat{k}) = -\sum_{k=1}^{K} P(z=k|w)\log P(z=k|w). \tag{5}$$

A uniformly distributed $P(z|w)$ gives the highest value of $H(\hat{k})$, whereas a salient and sparse $P(z|w)$ gives a small value of $H(\hat{k})$. Figure 5 shows the plot of $E(|\mathcal{M}|)$ and $H(\hat{k})$ for different values of $\mathcal{I}_{di}$. One can observe that as the entropy increases, the expectation value increases as well for large values of $\mathcal{I}_{di}$. Intuitively, when $P(z|w)$ is close to an uniform distribution (usually at the beginning, as topics are randomly initialized), it is more likely that sampled topics are different, thus $E(|\mathcal{M}|)$ is close to $\mathcal{I}_{di}$. As a consequence, a high sampling rate is preserved in the next iteration, which implies the solution space is not well explored yet. On the other hand, when $P(z|w)$ gets salient and sparse (usually after some iterations), sampling several times may result in duplicate topics, thus $E(|\mathcal{M}|)$ is smaller than $\mathcal{I}_{di}$. Therefore, a lower sampling rate is encouraged in the next iteration. Since $P(z|w)$ gradually becomes sparse in the training procedure, the dynamic sampling ensures a descending sampling rate and provides significant speedup. In conclusion, FastLDA, SparseLDA and ECGS-*Dynamic* all exploit the sparseness $P(z|w)$ to accelerate the standard CGS algorithm but from different perspectives.

The time complexity of ECGS-*Dynamic* is bounded by the complexity of ECGS-*Shortcut* (lower bound) and the standard CGS (upper bound). Notably, for every type in the document we first have to generate Γ_{di}, which needs an additional `multinomial`() operation. Therefore, for types that have $N_{di} \leqslant 2$, dynamic sampling can not give us any benefit on efficiency. In practice, we apply dynamic sampling only for w_{di} such that $N_{di} \geqslant 3$. For the

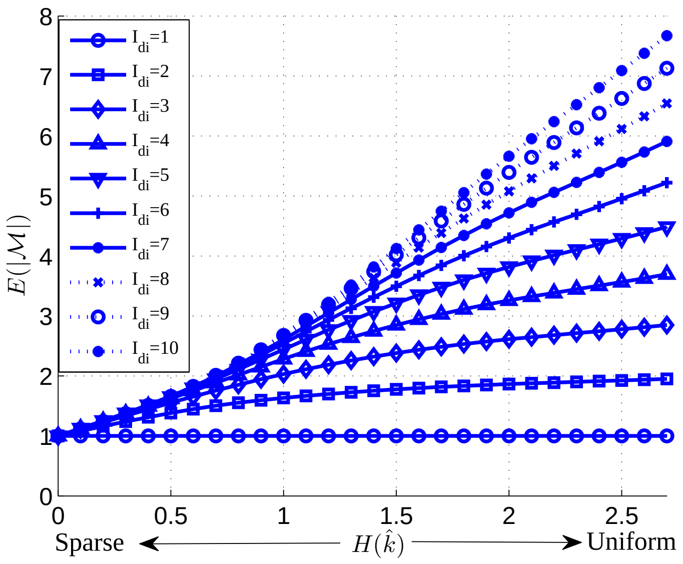


Figure 5: Expectation $E(|\mathcal{M}|)$ and entropy $H(\hat{k})$ for different of values $\mathcal{I}_{di}$. We use the entropy to measure the sparseness of $P(z|w)$. A high value of $H(\hat{k})$ indicates that $P(z|w)$ is close to an uniform distribution, whereas a low value of $H(\hat{k})$ indicates that $P(z|w)$ is a sparse distribution. Given a certain value of $\mathcal{I}_{di}$, a sparse distribution $P(z|w)$ results in low values of $E(|\mathcal{M}|)$. On contrary, an uniformly distributed $P(z|w)$ results in high values of $E(|\mathcal{M}|)$.

remaining types the standard CGS is conducted. The space complexity of ECGS-*Dynamic* is approximately two times of the standard CGS.

We summarize different CGS algorithms described above in Table 2 to highlight the characteristic of each algorithm.

Name	Characteristic	$\mathcal{I}_{di} =$	Reference
Standard CGS	No optimization	N_{di}	Fig.1
ECGS-Standard	Binary search and code-level optimization	N_{di}	Fig.1
ECGS-*Shortcut*	Sampling only one time for each type	1	Fig.2
ECGS-*Dynamic*	Sampling times is a random variable	$\mathcal{I}_{di} \sim P(\mathcal{I}_{di}\|\Gamma_{di})$	Fig.3

Table 2: Summary of differences between all CGS algorithms described in this paper. In our implementation, ECGS-*Shortcut* and ECGS-*Dynamic* is based on the code of ECGS-Standard, thus inherit all optimization tricks used in ECGS-Standard.

4.3 Parallelization

To further improve the efficiency on large-scale data set, we parallelize ECGS in AD-LDA fashion. As we do not modify the sampling formula itself, the only variable introduced in ECGS-*Dynamic* is the parameter vector of the multinomial distribution Γ. Since Γ is a document specific variable, there is no dependency between Γ_{di} and $\Gamma_{d'i}$ if $d \neq d'$. Thus, a local Γ^p can be maintained on processor p as well, without any communication or synchronization in between. To summarize, parallelizing ECGS is straight-forward and no special treatments are needed.

5. Experiments

The experiments are conducted on three data sets: KOS blog entries (from dailykos.com), NIPS full papers (from books.nips.cc), and NYTimes news articles (from ldc.upenn.edu)[7]. These three data sets span a wide range of collection size, content, and average document length. The KOS data set is the smallest one, NYTimes data set is relatively large, while NIPS data set is moderately sized. Therefore they are useful for fully demonstrating the performance of ECGS. For each collection, after tokenization and removal of stopwords, the vocabulary of unique words was truncated by only keeping words that occurred more than ten times. The size parameters for these three data sets are shown in Table 3.

Name	D	W	V	$\sum_d N_d/W$
KOS	$3,430$	0.4×10^6	$6,906$	0.755
NIPS	$1,500$	6.4×10^6	$12,419$	0.385
NYTimes	$300,000$	100×10^6	$102,660$	0.700

Table 3: Size parameters of the three data sets used in experiments. The value of $\sum_d N_d/W$ characterizes the redundancy of the data set. The smaller the value is, the more repetitive tokens exist in the data, thus the more redundancy in the data set. $\sum_d N_d/W$ also denotes the average sampling rate for ECGS-*Shortcut* .

The purpose of the experiments is to show the validity of the ECGS algorithms, especially, the ECGS-*Dynamic* , and to demonstrate the speedup against standard CGS. We present the experimental results from three perspectives. First, we use perplexity curve to validate the convergence of ECGS-*Dynamic* and ECGS-*Shortcut* on KOS and NIPS data sets. Second, we measure the execution time of ECGS-*Dynamic* , ECGS-*Shortcut* comparing with FastLDA and GibbsLDA implementations on all three data sets, and report their corresponding speedup. Finally, we examine the performance of the parallelized ECGS algorithm on NYT data set and compare it with the PLDA implementation. With the goal of repeatability, we have made our ECGS code publicly available[8].

Before explaining our experiments, we describe here the parameter specifications used to run our experiments. For all data sets, we run 500 iterations (including burn-in period) with

7. All data sets are available at `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words`
8. `http://home.in.tum.de/~xiaoh/pub/ecgs.html`

the empirical Dirichlet priors $\alpha = 50/K$ and $\beta = 0.02$ proposed by Griffiths and Steyvers (2004). The value of 500 iterations is chosen to guarantee that burn-in had occurred, and topics are drawn from the converged posterior distribution. For ECGS-*Dynamic* , the damping factor is set to $\gamma = 1$. In the perplexity experiment different values of γ are studied. All experiments are conducted with 4-fold cross-validation, each time with a different random initialization. Significant tests are done with t-test at the 5% significance level. All implementations are compiled using `gcc` with argument `-O3`, to enable the full optimization of the compiler and achieve the best performance. Experiments concerning time measurement are conducted on a Linux machine with 8 CPUs, each 2.7 Ghz and 64GB of memory in total.

5.1 Convergence Analysis

We first use perplexity to measure the convergence of ECGS-*Shortcut* and ECGS-*Dynamic* . Given test set D, the perplexity can be computed as follows:

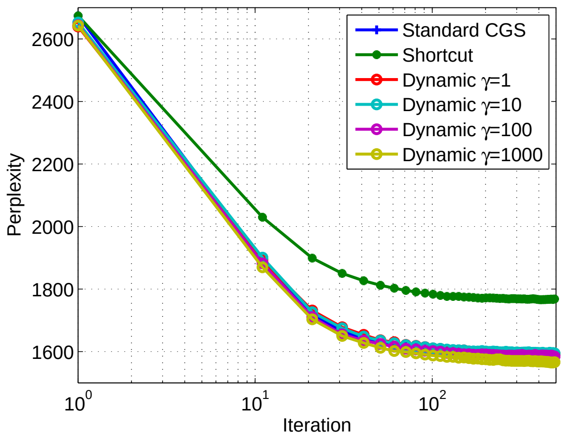
$$\text{Perplexity}(D) = \exp\left(-\frac{1}{N}\sum_{d=1}^{D}\sum_{i=1}^{N_d} N_{di}\log\sum_{k=1}^{K}\theta_{w_{di},k}\phi_{d,k}\right), \tag{6}$$

$$\text{where}\quad \theta_{v,k} = \frac{C_{vk}+\beta}{\sum_{v'} C_{v'k} + V\beta},\quad \phi_{d,k} = \frac{C_{dk}+\alpha}{\sum_{k'} C_{dk'} + K\alpha}.$$

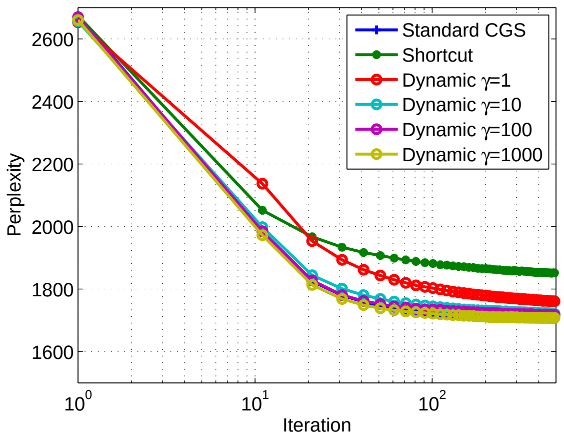
We report the average perplexity of 5 randomly initialized runs on KOS and NIPS data sets with $K = 40$. For each run, 3/4 data is used for training, 1/4 is used for testing. Perplexity as a function of the number of iterations for standard CGS, ECGS-*Shortcut* and ECGS-*Dynamic* algorithms is depicted in Figure 6. For ECGS-*Dynamic* , we also plot perplexity curves given by the algorithm with different damping factors γ and study the influence of γ on the model convergences. We treat the perplexity given by standard CGS as the ground-truth. One can observe, that ECGS-*Shortcut* converges to a suboptimal value on both data sets. This problem can be attributed to the cursory sampling strategy ignoring the repetitive tokens in the data set. Notably, ECGS-*Dynamic* converges to the ground truth as rapidly as standard CGS. On KOS data set, ECGS-*Dynamic* converges to the same perplexity result as standard CGS[9]. On NIPS data set, ECGS-*Dynamic* with $\gamma = 1$ converges to a slightly higher perplexity comparing to the ground-truth[10], while it is still significantly lower than the perplexity given by ECGS-*Shortcut* [11]. Although intuitively the choice of γ will surely affect the model convergence, the ECGS-*Dynamic* however seems to be insensitive to the γ according to this figure. To further examine the influence of γ in the ECGS-*Dynamic* algorithm, we depict the average sampling rate $\bar{\mathcal{S}}^t$ as a function of the number of iterations t in Figure 7. Figure 7 clearly illustrates two facts. First, $\bar{\mathcal{S}}^t$ always converges to a value (termed $\bar{\mathcal{S}}^\infty$) that lies between 1 (given by standard CGS) and $\sum_d N_d/W$ (given by ECGS-*Shortcut*). Second, the damping factor γ substantially affects the curve of $\bar{\mathcal{S}}^t$. The bigger γ is, the slower $\bar{\mathcal{S}}^t$ converges, and the larger the value of $\bar{\mathcal{S}}^\infty$ is. Since the average sampling rate is directly related to the running time of the algorithm, it is

9. Two-tailed test, p-value is 0.81
10. Right-tailed test, p-value is 7×10^{-4}
11. Left-tailed test, p-value is 4×10^{-7}

crucial to understand the factors that affects $\bar{S}^{\infty}$. The lower bound of $\bar{S}^{\infty}$ is determined by γ and $\sum_d N_d/W$. This implies that the speedup of ECGS-*Dynamic* may vary on different data sets.



(a) Perplexity curve on KOS data set.

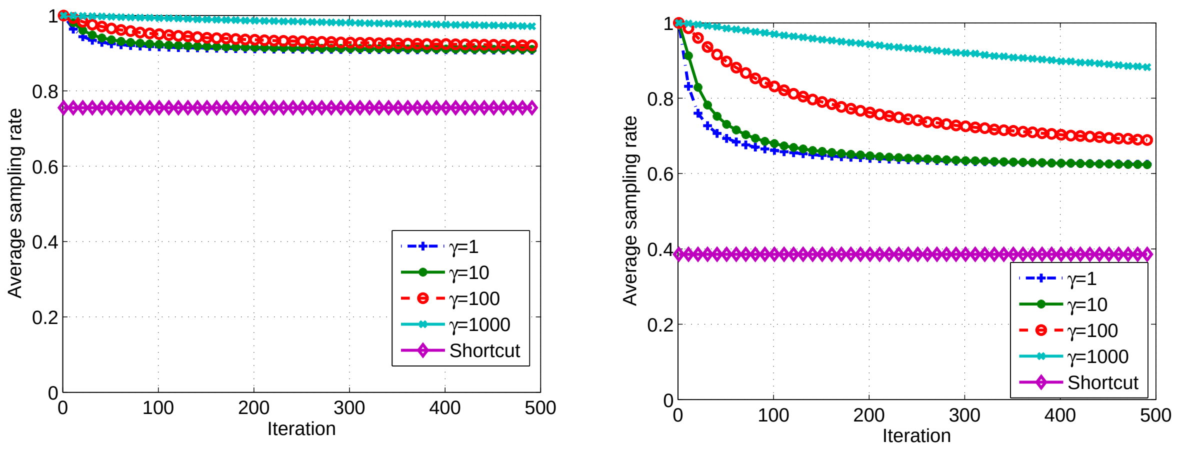


(b) Perplexity curve on NIPS data set.

Figure 6: Perplexity versus number of iterations (a small perplexity value is better). The models are trained on KOS data set (left) and NIPS data set (right) with 40 topics and 500 iterations. The result of ECGS-*Dynamic* with different damping factors $\gamma = 1, 10, 100, 1000$ is showed.

5.2 Sequential Speedup Results

The average runtime per iteration of the different CGS algorithms for different values of K is depicted in Figure 8. The baseline implementation is given by GibbsLDA, which employs standard CGS but without any code-level optimization. We use ECGS-Standard to denote the implementation based on GibbsLDA with included tricks such as binary search, loop fusion and other trivial optimizations. These tricks are also inherited in ECGS-*Dynamic* and ECGS-*Shortcut* implementations. In general, the speedup of all ECGS algorithms are remarkable on the three data sets. The ECGS-*Shortcut* enjoys the fastest speed among all algorithms, however it can not converge to the optimum. Besides, it is surprising to see that by just doing some code-level optimization, ECGS-*Shortcut* has 2 times lead over FastLDA. ECGS-*Dynamic* performs on average 4-6 times faster than GibbsLDA and 2 times faster than ECGS-Standard, meanwhile it converges to the equivalent model given by standard CGS. Another interesting fact is that the speedup of ECGS-*Dynamic* and ECGS-*Shortcut* is highly related to the redundancy of the data set. As summarized in Table 3, the redundancy of a data set is measured by $\sum_d N_d/W$. On a small data set like KOS, ECGS-*Dynamic* can not substantially reduce the sampling times as there are not many repetitive tokens in each document. When the data set is large and span a wide range of content, like NYT, the vocabulary size is usually large as well. Consequently, less redundancy exists in the data set, and thus ECGS-*Dynamic* also fails to show a promising speedup. Despite these



(a) Average sampling rate curves on KOS. The average sampling rate of ECGS-*Shortcut* is 0.755.

(b) Average sampling rate curves on NIPS. The average sampling rate of ECGS-*Shortcut* is 0.385.

Figure 7: Average sampling rate of ECGS-*Dynamic* with different damping factors γ versus number of iterations. The models are trained on KOS data set (left) and NIPS data set (right) with 40 topics and 500 iterations. The constant average sampling rate of ECGS-*Shortcut* is depicted for comparison. As the average sampling rate and speedup are in inverse proportion, one can obtain speedup curves with different γ by flipping these figures.

two cases, when the data set is large but the content is concentrated in few areas, ECGS-*Dynamic* scales well and demonstrates a full speedup (e.g. NIPS data set).

5.3 Parallel Speedup Results

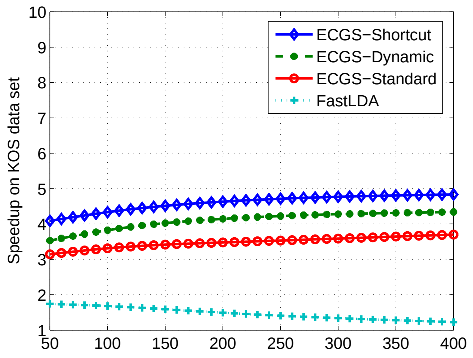
Finally we demonstrate the performance of the parallel algorithm. Figure 9 shows the execution time of the parallelized ECGS algorithms against PLDA on NIPS and NYT data set. In general, by increasing the number of processors, we can significantly reduce the training time. Indeed, on the large-scale data set like NYT, all parallel implementations can achieve approximately linear speedup of 7 on up to 8 processors. As PLDA is based on the standard CGS, the actual execution time is much higher than of the parallel ECGS. By joining optimization and parallelization together, ECGS-*Dynamic* dramatically reduces the training time of 500 iterations on NYT from 5 days (given by sequential standard CGS) to 5 hours without losing optimality.

6. Conclusion

Collapsed Gibbs sampling is a method which can be applied to approximate intractable integrals in probabilistic generative models such as LDA. The method however, suffers of high computational complexity and is therefore limited applicable on large data sets. To overcome this limitation, we proposed a novel and efficient Collapsed Gibbs sampling strat-

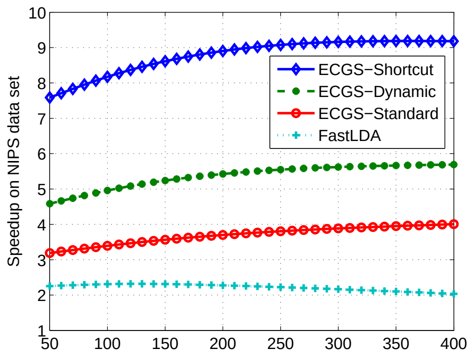
Algorithm	50	100	200	400
ECGS-Standard	0.2	0.3	0.6	1.1
ECGS-*Dynamic*	0.1	0.3	0.5	0.9
ECGS-*Shortcut*	0.1	0.2	0.4	0.8
FastLDA	0.3	0.6	1.4	3.2
GibbsLDA	0.5	1.0	2.0	3.9

(a) KOS data set



Algorithm	50	100	200	400
ECGS-Standard	0.7	1.2	2.2	4.0
ECGS-*Dynamic*	0.5	0.8	1.5	2.8
ECGS-*Shortcut*	0.3	0.5	0.9	1.7
FastLDA	1.0	1.8	3.6	7.9
GibbsLDA	2.2	4.2	8.1	16.0

(b) NIPS data set



Algorithm	50	100	200	400
ECGS-Standard	42.0	67.1	122.7	220.0
ECGS-*Dynamic*	37.3	60.6	104.6	194.4
ECGS-*Shortcut*	32.6	52.8	88.3	164.7
FastLDA	52.0	111.5	251.5	540.0
GibbsLDA	128.3	243.0	482.4	936.3

(c) NYT data set

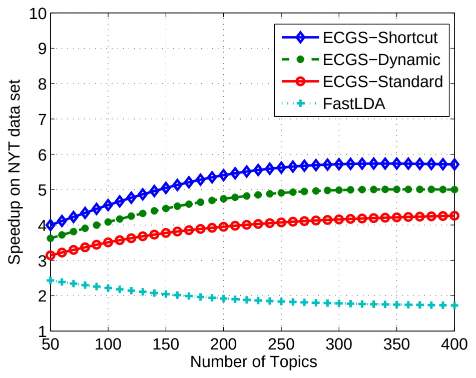
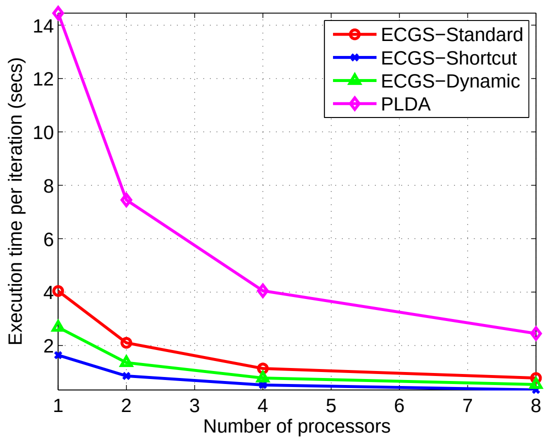
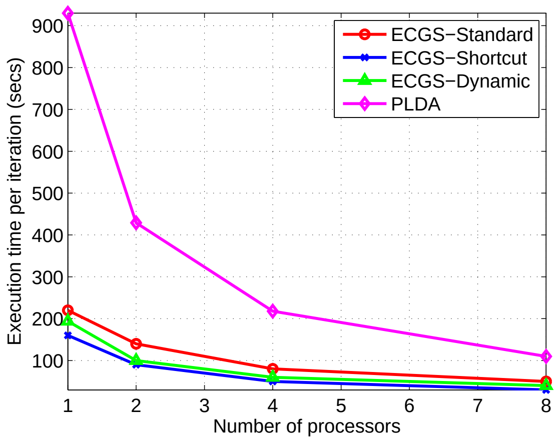


Figure 8: Runtime results of the sequential algorithms for data sets KOS, NIPS and NYT. All algorithms are trained with 500 iterations. The tables show the average runtime per iteration (in seconds) as a function of the number of topics, which is calculated by the total runtime/500. The figures illustrate the speedup as a function of the number of topics of the corresponding data sets. The speedup is calculated as the average runtime of GibbsLDA divided by average runtime of the benchmark algorithms.

egy: *dynamic sampling*. Unlike previous works, our strategy focus on reducing the sampling times for each word. In particular, the dynamic sampling method significantly increased the sampling speed while retaining all the optimality guarantees associated with standard Gibbs sampling. Additionally, we presented an overview of already proposed techniques



(a) Average execution time on NIPS.



(b) Average execution time on NYT.

Figure 9: Average execution time per iteration on NIPS dataset (left) and NYT data set (right). The parallel ECGS is implemented using OpenMP, PLDA is implemented using MPI. The models are trained with 400 topics for 500 iterations.

and conducted a comparison study. The experiments on three public data sets showed that our strategy entertain promising speedups. Moreover, we proposed a parallelization of the strategy and demonstrated a significant speedup (training of 5 days reduced to 5 hours). It has not escaped our mind that there is still some room to further improve the efficiency of Gibbs sampling, namely, by combining FastLDA and ECGS.

Acknowledgments

We thank Ping Luo, Zhiyong Shen for their advices on the previous draft. We also thank Hosam M. Mahmoud for his derivation of the urn problem, which enables us to develop deeper understanding of the dynamic sampling procedure.

Appendix

Proof of (4)

We formulate the dynamic sampling procedure in Figure 3 line 5 to line 11 as the following urn problem. Given an urn with an infinite number of balls, divided up among K colors. Color k has proportion P_k of the total balls, and $\sum_{k=1}^{K} P_k = 1$. We draw n balls out of the urn with replacement. Let m be the number of different colors in the drawn sequence.

For color k, the probability that at least one of the balls drawn has that color is $1 - (1 - P_k)^n$. The expectation of having m different colors is hence the sum over K colors

$$E(m) = \sum_{k=1}^{K} \big(1 - (1 - P_k)^n\big). \tag{7}$$

References

D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

K.R. Canini, L. Shi, and T.L. Griffiths. Online inference of topics with latent Dirichlet allocation. In *AISTATS*, volume 5, 2009.

T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228, 2004.

T. Minka and J. Lafferty. Expectation-propagation for the generative aspect model. In *UAI*, pages 352–359, 2002.

D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for latent dirichlet allocation. volume 20, pages 1081–1088, 2007.

I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *SIGKDD*, pages 569–577. ACM, 2008.

Y. Wang, H. Bai, M. Stanton, W.Y. Chen, and E. Chang. Plda: Parallel latent dirichlet allocation for large-scale applications. In *AAIM*, pages 301–314, 2009.

Feng Yan, Ningyi Xu, and Yuan Qi. Parallel inference for latent dirichlet allocation on graphics processing units. In *NIPS*, volume 20, pages 2134–2142, 2009.

L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *SIGKDD*, pages 937–946. ACM, 2009.