Reducing the Sampling Complexity of Topic Models

Aaron Q. Li
CMU Language Technologies
Pittsburgh, PA
aaronli@cmu.edu

Amr Ahmed
Google Strategic Technologies
Mountain View, CA
amra@google.com

Sujith Ravi
Google Strategic Technologies
Mountain View, CA
sravi@google.com

Alexander J. Smola
CMU MLD and Google ST
Pittsburgh PA
alex@smola.org

ABSTRACT

Inference in topic models typically involves a sampling step to associate latent variables with observations. Unfortunately the generative model loses sparsity as the amount of data increases, requiring $O(k)$ operations per word for k topics. In this paper we propose an algorithm which scales linearly with the number of actually instantiated topics k_d in the document. For large document collections and in structured hierarchical models $k_d \ll k$. This yields an order of magnitude speedup. Our method applies to a wide variety of statistical models such as PDP [16, 4] and HDP [19].

At its core is the idea that dense, slowly changing distributions can be approximated efficiently by the combination of a Metropolis-Hastings step, use of sparsity, and amortized constant time sampling via Walker's alias method.

Keywords

Sampling; Scalability; Topic Models; Alias Method

1. INTRODUCTION

Topic models are some of the most versatile tools for modeling statistical dependencies. Given a set of observations $x_i \in \mathcal{X}$, such as documents, logs of user activity, or communications patterns, we want to infer the hidden causes motivating this behavior. A key property in topic models is that they model $p(x)$ via a discrete hidden factor, z via $p(x|z)$ and $p(z)$. For instance, z may be the cluster of a document. In this case it leads to Gaussian and Dirichlet mixture models [14]. When z is a vector of topics associated with individual words, this leads to Latent Dirichlet Allocation [3]. Likewise, whenever z indicates a term in a hierarchy, it leads to structured and mixed-content annotations [19, 2, 4, 12].

1.1 Sparsity in Topic Models

One of the key obstacles in performing scalable inference is to draw $p(z|x)$ from the discrete state distribution associated with the data. A substantial improvement in this context was provided by [22] who exploited sparsity to decompose the collapsed sampler [9] for Latent Dirichlet Allocation. As a result the sampling cost can be reduced from $O(k)$, the total number of topics to $O(k_d + k_w)$, i.e. the number k_w of topics occurring for a particular word w and k_d for a particular document d. This insight led to an order of magnitude improvement for sampling topic models, thus making their implementation feasible at a large scale. In fact, the strategy is sufficiently robust that it can be extended to settings where the topic smoother depends on the words [15].

For small datasets the assumption $k_d + k_w \ll k$ is well satisfied. Unfortunately, as the number of documents grows, so does the number of topics in which a particular word occurs. In particular $k_w \to k$, since the probability of observing any particular topic for a given word is rarely nonzero: Assume that the probability of occurrence for a given topic for a word is bounded from below by δ. Then the probability of the topic occurring at least once in a collection of n documents is given by

$$1 - (1 - \delta)^n \geq 1 - e^{-n\delta} \to 1 \text{ for } n \to \infty.$$

From this it follows that $k_w = O(k)$ for $n = O(\delta^{-1} \log k)$. In other words, for large numbers documents the efficiencies discussed in [22] vanish. This is troubling, since in many industrial settings n can be in the order of billions to trillions. Consequently, with increasing amounts of data, the *time to process individual documents* increases due to loss of sparsity, thus leading to a *superlinear* increase in runtime.

On the other hand, the topic-sparsity for a given document essentially remains unchanged, regardless of the total number of related documents that are available. This is due to the fact that the number of tokens per document is typically less than $O(k)$. For instance, microblogs contain only dozens of words, yet admit to thousands of topics.[1] This situation is exacerbated when it comes to hierarchical and structured topic models, since there the number of (sub)topics can grow considerably more rapidly. Hence the use of sparsity is crucial in designing efficient samplers.

1.2 Metropolis-Hastings-Walker Sampling

The present paper proposes a new decomposition of the collapsed conditional probability, in conjunction with a

[1]Obviously, this approach would not work to infer topics for Dostojevski's *War and Peace*. That said, a plain topic model is an unlikely candidate to represent very long documents.

KDD'14, August 24–27, 2014, New York, NY, USA.

ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623756.

Metropolis-Hastings [7] scheme and the use of the alias method, introduced by Walker [20, 13], to amortize dense updates for random variables. This method is highly versatile. It defers corrections to the model and avoids renormalization. This allows us to apply it to both flat and hierarchical models. Experimental evaluation demonstrates the efficacy of our approach, yielding orders of magnitude acceleration and a simplified algorithm.

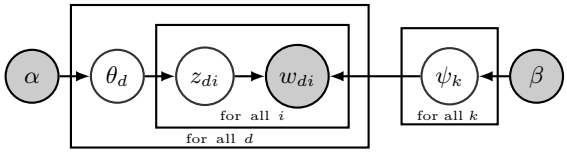
While we introduce our algorithm in the context of topic models, it is entirely general and applies to a much richer class of models. At its heart lies the insight that in many inference problems the model parameters only change relatively slowly during sampling. For instance, the location of cluster centers, the definition of topics, or the shape of autoregressive functions, only change relatively slowly. Hence, if we could draw from a distribution over k outcomes k times, Walker's alias method would allow us to generate samples in amortized constant time. At the same time, the Metropolis Hastings algorithm allows us to use approximations of the correct probability distribution, provided that we compute ratios between successive states correctly. Our approach is to draw from the stale distribution in constant time and to accept the transition based on the ratio between successive states. This step takes constant time. Moreover, the proposal is independent of the current state. Once k samples have been drawn, we simply update the alias table. In honor of the constitutent algorithms we refer to our technique as the Metropolis Hastings Walker (MHW) sampler.

2. TOPIC MODELS

We begin with a brief introduction to topic models and the associated inference problems. This includes a short motivation of sampling schemes in the context collapsed samplers [9, 18] and of stochastic variational models [21]. It is followed by a description of extensions to hierarchical models.

2.1 Latent Dirichlet Allocation

In LDA [3] one assumes that documents are mixture distributions of language models associated with individual topics. That is, the documents are generated following the graphical model below:



For each document d draw a topic distribution θ_d from a Dirichlet distribution with concentration parameter α

$$\theta_d \sim \text{Dir}(\alpha). \tag{1}$$

For each topic t draw a word distribution from a Dirichlet distribution with concentration parameter β

$$\psi_t \sim \text{Dir}(\beta). \tag{2}$$

For each word $i \in \{1 \dots n_d\}$ in document d draw a topic from the multinomial θ_d via

$$z_{di} \sim \text{Discrete}(\theta_d). \tag{3}$$

Draw a word from the multinomial $\psi_{z_{di}}$ via

$$w_{di} \sim \text{Discrete}(\psi_{z_{di}}). \tag{4}$$

The beauty of the Dirichlet-multinomial design is that the distributions are conjugate. This means that the multinomial distributions θ_d and ψ_k can be integrated out, thus allowing one to express $p(w, z|\alpha, \beta, n_d)$ in closed-form [9]. This yields a Gibbs sampler to draw $p(z_{di}|\text{rest})$ efficiently. The conditional probability is given by

$$p(z_{di}|\text{rest}) \propto \frac{(n_{td}^{-di} + \alpha_t)(n_{tw}^{-di} + \beta_w)}{n_t^{-di} + \bar{\beta}}. \tag{5}$$

Here the count variables n_{td}, n_{tw} and n_t denote the number of occurrences of a particular (topic,document) and (topic,word) pair, or of a particular topic respectively. Moreover, the superscript $\cdot^{-di}$ denotes said count when ignoring the pair (z_{di}, w_{di}). For instance, n_{tw}^{-di} is obtained when ignoring the (topic,word) combination at position (d, i). Finally, $\bar{\beta} := \sum_w \beta_w$ denotes the joint normalization.

At first glance, sampling from (5) appears to cost $O(k)$ time since we have k nonzero terms in a sum that needs to be normalized. [22] devised an ingenious strategy for exploiting sparsity by decomposing terms into

$$p(z_{di}|\text{rest}) \propto \beta_w \frac{\alpha_t}{n_t^{-di} + \bar{\beta}} + n_{td}^{-di} \frac{\beta_w}{n_t^{-di} + \bar{\beta}} + n_{tw}^{-di} \frac{n_{td}^{-di} + \alpha_t}{n_t^{-di} + \bar{\beta}}$$

As can be seen, for small collections of documents only the first term is dense, and more specifically, $\sum_t \alpha_t/(n_t^{-di} + \bar{\beta})$ can be computed from $\sum_t \alpha_t/(n_t + \bar{\beta})$ in $O(1)$ time. That is, whenever both n_{td} and n_{tw} are sparse, sampling from $p(z_{di}|\text{rest})$ can be accomplished efficiently. The use of packed index variables and a clever reordering of (topic,count) pairs further improve efficient sampling to $O(k_w + k_d)$.

Stochastic variational inference [11] requires an analogous sampling step. The main difference being that rather than using $\frac{n_{tw}+\beta_w}{n_t+\bar{\beta}}$ to capture $p(w|t)$ one uses a natural parameter η_{tw} associated with the conjugate variational distribution. Unfortunately this renders the model dense, unless rather careful precautions are undertaken [11] to separate residual dense and sparse components.

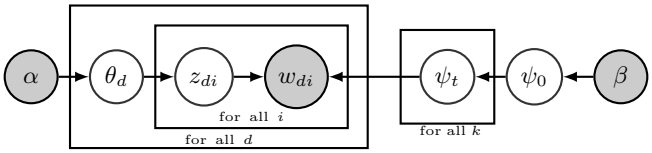
Instead, we devise a sampler to draw from $p(z_{di}|\text{rest})$ in amortized $O(k_d)$ time. We accomplish this by using

$$p(z_{di}|\text{rest}) \propto n_{td}^{-di} \frac{n_{tw}^{-di} + \beta_w}{n_t^{-di} + \bar{\beta}} + \frac{\alpha_t(n_{tw}^{-di} + \beta_w)}{n_t^{-di} + \bar{\beta}} \tag{6}$$

Here the first term is sparse in k_d and we can draw from it in $O(k_d)$ time. The second term is dense, regardless of the number of documents (this holds true for stochastic variational samplers, too). However, the 'language model' $p(w|t)$ does not change too drastically whenever we resample a single word. The number of words is huge, hence the amount of change per word is concomitantly small. This insight forms the basis for applying Metropolis-Hastings-Walker sampling.

2.2 Poisson Dirichlet Process

To illustrate the fact that the MHW sampler also works with models containing a dense generative part, we describe its application to the Poisson Dirichlet Process [4, 16]. The model is given by the following variant of the LDA model:



In a conventional topic model the language model is simply given by a multinomial draw from a Dirichlet distribution. This fails to exploit distribution information between topics, such as the fact that all topics have the same common underlying language. A means for addressing this problem is to add a level of hierarchy to model the distribution over ψ_t via $\prod_t p(\psi_t|\psi_0)p(\psi_0|\beta)$ rather than $\prod_t p(\psi_t|\beta)$. Such a model is depicted above.

The ingredients for a refined language model are a Pitman-Yor Topic Model (PYTM) [17] that is more appropriate to deal with natural languages. This is then combined with the Poisson Dirichlet Process (PDP) [16, 4] to capture the fact that the number of occurences of a word in a natural language corpus follows power-law. Within a corpus, the frequency of a word is approximately inversely proportional to its ranking in number of occurences. Each draw from a Poisson Dirichlet Process PDP(b, a, ψ_0) is a probability distribution. The base distribution ψ_0 defines the common underlying distribution shared across the generated distributions. Under the settings of Pitman-Yor Topic Model, each topic defines a distribution over words, and the base distribution defines the common underlying common language model shared by the topics. The concentration parameter b controls how likely a word is to occur again while being sampled from the generated distribution. The discount parameter a prevents a word to be sampled too often by imposing a penalty on its probability based on its frequency. The combined model described explicityly in [5]:

$$\begin{aligned} \theta_d &\sim \text{Dir}(\alpha) & \psi_0 &\sim \text{Dir}(\beta) \\ z_{di} &\sim \text{Discrete}(\theta_d) & \psi_t &\sim \text{PDP}(b, a, \psi_0) \\ w_{di} &\sim \text{Discrete}\left(\psi_{z_{di}}\right) \end{aligned}$$

As can be seen, the document-specific part is identical to LDA whereas the language model is rather more sophisticated. Likewise, the collapsed inference scheme is analogous to a Chinese Restaurant Process [6, 5]. The technical difficulty arises from the fact that we are dealing with distributions over countable domains. Hence, we need to keep track of multiplicities, i.e. whether any given token is drawn from β_i or β_0. This will require the introduction of additional count variables in the collapsed inference algorithm.

Each topic is equivalent to a restaurant. Each token in the document is equivalent to a customer. Each type of word corresponds each type of dish served by the restaurant. The same results in [6] can be used to derive the conditional probability by introducing axillary variables:

- s_{tw} denotes the number of tables serving dish w in restaurant t. Here t is the equivalent of a topic.
- r_{di} indicates whether w_{di} opens a new table in the restaurant or not (to deal with multiplicities).
- m_{tw} denotes the number of times dish w has been served in restaurant t (analogously to n_{wk} in LDA).

The conditional probability is given by:

$$p(z_{di} = t, r_{di} = 0|\text{rest}) \propto \frac{\alpha_t + n_{dt}}{b_t + m_t} \frac{m_{tw} + 1 - s_{tw}}{m_{tw} + 1} \frac{S^{m_{tw}+1}_{s_{tw},a_t}}{S^{m_{tw}}_{s_{tw},a_t}} \quad (7)$$

if no additional 'table' is opened by word w_{di}. Otherwise

$$\begin{aligned} & p(z_{di} = t, r_{di} = 1|\text{rest}) \quad (8) \\ \propto & (\alpha_t + n_{dt}) \frac{b_t + a_t s_t}{b_t + m_t} \frac{s_{tw} + 1}{m_{tw} + 1} \frac{\gamma + s_{tw}}{\bar{\gamma} + s_t} \frac{S^{m_{tw}+1}_{s_{tw}+1,a_t}}{S^{m_{tw}}_{s_{tw},a_t}} \end{aligned}$$

Here $S^N_{M,a}$ is the generalized Stirling number. It is given by

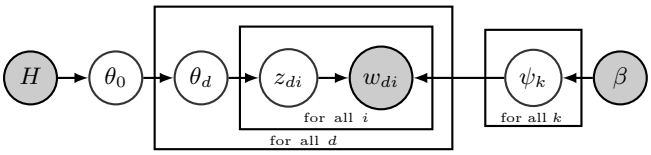
$$S^{N+1}_{M,a} = S^N_{M-1,a} + (N - Ma)S^N_{M,a} \text{ and } S^N_{M,a} = 0$$

for $M > N$, and $S^N_{0,a} = \delta_{N,0}$. A detailed analysis is given in [4]. Moreover we have $m_t = \sum_w m_{tw}$, and $s_t = \sum_t s_{tw}$.

Similar to the conditional probability expression in LDA, these two expressions can be written as a combination of a sparse term and a dense term, simply by splitting the factor $(\alpha_t + n_{dt})$ into its sparse component n_{dt} and its dense counterpart α_t. Hence we can apply the same strategy as before when sampling topics from LDA, albeit now using a twice as large space of state variables.

2.3 Hierarchical Dirichlet Process

To illustrate the efficacy and generality of our approach we discuss a third case where the *document* model itself is more sophisticated than a simple collapsed Dirichlet-multinomial. We demonstrate that there, too, inference can be performed efficiently. Consider the two-level topic model based on the Hierarchical Dirichlet Process [19] (HDP-LDA). In it, the topic distribution for each document θ_d is drawn from a Dirichlet process DP(b_1, θ_0). In turn, θ_0 is drawn from a Dirichlet process DP$(b_0, H(\cdot))$ governing the distribution over topics. In other words, we add an extra level of hierarchy on the document side (compared to the extra hierarchy on the language model used in the PDP).



More formally, the joint distribution is as follows:

$$\begin{aligned} \theta_0 &\sim \text{DP}(b_0, H(\cdot)) & \psi_t &\sim \text{Dir}(\beta) \\ \theta_d &\sim \text{DP}(b_1, \theta_0) \\ z_{di} &\sim \text{Discrete}(\theta_d) \\ w_{di} &\sim \text{Discrete}\left(\psi_{z_{di}}\right) \end{aligned}$$

By construction, DP$(b_0, H(\cdot))$ is a Dirichlet Process, equivalent to a Poisson Dirichlet Process PDP$(b_0, a, H(\cdot))$ with the discount parameter a set to 0. The base distribution $H(.)$ is often assumed to be a uniform distribution in most cases.

At first, a base θ_0 is drawn from DP$(b_0, H(\cdot))$. This governs how many topics there are in general, and what their overall prevalence is. The latter is then used in the next level of the hierarchy to draw a document-specific distribution θ_d that serves the same role as in LDA. The main difference is that unlike in LDA, we use θ_0 to infer which topics are more popular than others.

It is also possible to extend the model to more than two levels of hierarchy, such as the infinite mixture model [19]. Similar to Poisson Dirichlet Process, an equivalent Chinese Restaurant Franchise analogy [6, 19] exists for Hierarchical Dirichlet Process with multiple levels. In this analogy, each Dirichlet Process is mapped to a single Chinese Restau-

rant Process, and the hierarchical structure is constructed to identify the parent and children of each restaurant.

The general (collapsed) structure is as follows: let N_j be the total number of customers in restaurant j and n_{jt} be the number of customers in restaurant j served with dish t. When a new customer (a token) enters restaurant j with the corresponding Dirichlet Process $DP(b_j, H_j(\cdot))$, there are two types of seating arrangement for the customer:

- With probability $\frac{n_{jt}}{b_j+N_j}$ the customer is served with dish (topic) t and sits at an existing table.
- With probability $\frac{b_j}{b_j+N_j}$ the customer sits at a new table served with a new dish t drawn from $H_j(\cdot)$.

In the event that the customer sits at a new table, a phantom customer is sent upstream the hierarchy to the parent restaurant of j, denoted by j', with corresponding Dirichlet Process $DP(b_{j'}, H_{j'}(\cdot))$. The parent restaurant then decides the seating arrangement of the phantom customer under the same rules. This process repeats, until there is no more parent restaurant or any of phantom customer decides to sit in an existing table in any parent restaurant along the path.

We use the block Gibbs sampler given in [6] as it allows us to extend our approach for multi-level Hierarchical Dirichlet Process, and performs better than the samplers given in [19] and the collapsed Gibbs sampler given in [4], as measured in convergence speed, running time, and topic quality.

The key difference of [6] relative to [19] is that rather than keeping track of relative assignments of tables to each other (and the resulting multiplicities and infrequent block moves) it simply keeps track of the level within the hierarchy of restaurants at which an individual customer opens a new table. The advantage is that this allows us to factor out the relative assignment of customers to specific tables but rather only keep track of the dishes that they consume. The obvious downside being that a small number of customers can be blocked from moves if they opened a table at a high position of the hierarchy that other customers depend upon. Improving mixing in this context is subject to future work.

In the setting studied above we only have a two-level HDP: that of the parent DP tying all documents together and the DP within each document, governing its topic distribution. We use $z_{di} \in \mathbb{N}$ to denote the topic indicator of word i at position d and $u_{di} \in \{0, 1\}$ to indicate whether a new table is opened at the root level (i.e. $u_{di} = 1$). Moreover, define s_{td} to be the table counter for document d, i.e. the number of times a table serving topic t has been opened, and let s_t be the associated counter for the base DP, associated with tables opened at the parent level. Finally, let $s := \sum_t s_t$ be the total number of tables opened at the root level.

Clearly the situation where $s_t = 0$ and $u_{di} = 0$ is impossible since this would imply that we are opening a new table at document d while there is no matching table available at the root level. Hence for the collapsed sampler we only need to consider the following cases:

- A new root topic is generated. That is, we currently have s_t and need to set $u_{di} = 1$.
- A new table is added at document d. In this case we require that s_t, i.e. that the topic exists at the root level. Moreover, obviously it requires that $s_{td} = 0$ since we are adding the first table to serve dish t.
- No additional topic is introduced but we may be introducing an additional table.

This amounts to the following (unnormalized) conditional probabilities. See [6] for further details.

$$p\left(z_{di} = t, u_{di} = u | \text{rest}\right) \tag{9}$$
$$\propto \frac{\beta_w + m_{tw}}{\bar{\beta} + m_t} \begin{cases} \frac{b_0 b_1}{b_0+s} & \text{if } s_t = 0 \text{ and } u_{di} = 1 \\ \frac{b_1 s_t^2}{(s_t+1)(s+b_0)} & \text{if } s_t \neq 0 \text{ and } s_{td} = 0 \\ \frac{S^{n_{dt}+1}_{s_{dt}+1,0}}{S^{n_{dt}}_{s_{dt},0}} \frac{s_{dt}+1}{n_{dk}+1} & \text{if } s_t \neq 0 \text{ and } s_{td} \neq 0 \end{cases}$$

Expressions for the generalized form are analogous. Both forms contain a fraction with its numerator being the sum of a sparse term m_{tw} and a dense term β_w. Therefore, the conditional probability can be decomposed to a dense term multiplied by β_w, and a sparse term multiplied by m_{tw}. Applying the same methodology, the sampling complexity of a multi-level HDP can be reduced to $O(k_w)$.

3. METROPOLIS-HASTINGS-WALKER

We now introduce the key components for the MHW algorithm and how to use it in sampling topics. They consist of the alias method [20, 13] and a simplified version of the Metropolis-Hastings sampler [7].

3.1 Walker's Alias Method

Typically, when drawing from a distribution over l outcomes, it is accepted that one would need to perform $O(l)$ work to generate a sample. In fact, this is a lower bound, since we need to inspect each probability at least once before we can construct the sampler. However, what is commonly overlooked is that there exist algorithms that allow us to draw *subsequent samples from the same distribution* in $O(1)$ time. This means that drawing l samples from a distribution over l outcomes can be accomplished in $O(1)$ amortized time per draw. We make extensive use of this fact.

Denote by p_i with $i \in \{1 \dots l\}$ the probabilities of a distribution over l outcomes from which we would like to sample. The algorithm works by decomposing a distribution over l events into l bins of equal probability by pairing at most two events per bin. Since it 'robs' from the probabilities $p_i > 1/l$ and adds to those with $p_i < 1/l$ it is also referred to as 'Robin Hood' method [13]. The algorithm proceeds as follows:

GenerateAlias(p, l)
Initialize $L = H = \emptyset$ and $A = []$.
for $i = 1$ **to** l **do**
  if $p_i \leq l^{-1}$ **then**
    $L \leftarrow L \cup \{(i, p_i)\}$
  else
    $H \leftarrow H \cup \{(i, p_i)\}$
  end if
end for
while $L \neq \emptyset$ **do**
  Extract (i, p_i) from L and (h, p_h) from H
  $A \leftarrow [A, (i, h, p_i)]$
  if $p_h - p_i > l^{-1}$ **then**
    $H \leftarrow H \cup \{(h, p_h - p_i)\}$
  else
    $L \leftarrow L \cup \{(h, p_h - p_i)\}$
  end if
end while
return A

This yields an array A containing triples (i, h, p_h) with $p_h < l^{-1}$. It runs in $O(l)$ time since at each step one event is removed from the list. And the probabilities remain unchanged, as can be seen by induction. All we need to do now is to draw a random element from A and flip a biased coin to accept h or i with probability lp_h and $1 - lp_h$ respectively.

SampleAlias(A, l)
bin = RandInt(l)
$(i, h, p) = A$[bin]
if $lp >$ RandUnif(1) **then**
  return h
else
  return i
end if

Note that the alias method works since we are implicitly exploiting parallelism inherent in CPUs: as long as l does not exceed 2^{64} are guaranteed that even an information theoretically inefficient code will not require more than 64 bit, which can be generated in constant time.

3.2 Sampling with Proposal Distributions

Whenever we draw l identical samples from p it is clear that the above algorithm provides an $O(1)$ sampler. However, if p changes, it is difficult to apply the alias sampler directly. To address this, we use rejection sampling and Metropolis-Hastings procedures. Rejection sampling proceeds as follows:

Rejection(p, q, c)
repeat
  Draw $i \sim q(i)$
until $p(i) \geq cq(i)$RandUnif(1)
return i

Here p is the distribution we would like to draw from, q is a reference distribution that makes sampling easy, and $c \geq 1$ is chosen such that $cq(i) \geq p(i)$ for all i. We then accept with probability $\frac{p(i)}{cq(i)}$. It is well known that the expected number of samples to draw via **Rejection**(p, q, c) is c, provided that a good bound c exists. In this case we have the following:

Lemma 1 *Given l distributions p_i and q over l outcomes satisfying $c_i q \geq p_i$, the expected amortized runtime complexity for drawing using **SampleAlias**(A, l) and rejecting using **Rejection**(p_i, q, c_i) is given by $O\left(\frac{1}{l}\sum_{i=1}^{l} c_i\right)$.*

Proof. Preprocessing costs amortized $O(1)$ time. Each rejection sampler costs $O(c_i)$ work. Averaging over the draws proves the claim. □

In many cases, unfortunately, we do not know c_i, or computing c_i is essentially as costly as drawing from p_i itself. Moreover, in some cases c_i may be unreasonably large. In this situation we resort to Metropolis Hastings sampling [7] using a stationary proposal distribution. As in rejection sampling, we use a proposal distribution q and correct the effect of sampling from the 'wrong' distribution by a subsequent acceptance step. The main difference is that Metropolis Hastings can be considerably more efficient than Rejection sampling since it only requires that the ratios of probabilities are close rather than requiring knowledge of a uniform upper bound on the ratio. The drawback is that instead of drawing *iid* samples from p we end up with a *chain* of *dependent* samples from p, as governed by q.

For the purpose of the current method we only need to concern ourselves with stationary distributions p and q, i.e. $p(i) = p(i|j)$ and $q(i) = q(i|j)$, hence we only discuss this special case below. For a more general discussion see e.g. [8].

StationaryMetropolisHastings(p, q, n)
if no initial state exists **then** $i \sim q(i)$
for $l = 1$ **to** n **do**
  Draw $j \sim q(j)$
  if RandUnif(1) $< \min\left(1, \frac{p(j)q(i)}{p(i)q(j)}\right)$ **then**
    $i \leftarrow j$
  end if
end for
return i

As a result, provided that p and q are sufficiently similar, the sampler accepts most of the time. This is the case, e.g. whenever we use a stale variant of p as the proposal q. Obviously, a necessary requirement is that $q(i) > 0$ whenever $p(i) > 0$, which holds, e.g. whenever we incorporate a smoother.

3.3 MHW Sampling

In combining both methods we arrive at, what we believe is a significant improvement over each component individually. It works as follows:

Initialize $A \leftarrow$ **GenerateAlias**(p, l)
for $i = 1$ **to** $\frac{N}{n}$ **do**
  Update q as needed
  Sample $j \sim$ **StationaryMetropolisHastings**(p, A, n)
end for

Provided that the sampler mixes within n rounds of the MH-procedure, this generates draws from up-to-date versions of p. Note that a further improvement is possible whenever we can start with a more up-to-date draw from p, e.g. in the case of revisiting a document in a topic model. After burn-in the previous topic assignment for a given word is likely to be still pertinent for the current sampling pass.

Lemma 2 *If the Metropolis Hastings sampler over N outcomes using q instead of p mixes well in n steps, the amortized cost of drawing n samples from q is $O(n)$ per sample.*

This follows directly from the construction of the sampler and the fact that we can amortize generating the alias table. Note that by choosing a good starting point and after burn-in we can effectively set $n = 1$.

4. APPLICATIONS

We now have all components necessary for an accelerated sampler. The trick is to recycle old values for $p(w_{di}|z_{di})$ even when they change slightly and then to correct this via a Metropolis-Hastings scheme. Since the values change only slightly, we can therefore amortize the values efficiently. We begin by discussing this for the case of 'flat' topic models and extend it to hierarchical models subsequently.

4.1 Sampler for LDA

We now design a proposal distribution for (6). It involves computing the document-specific sparse term exactly and approximating the remainder with slightly stale data. Furthermore, to avoid the need to store a stale alias table A, we simply *draw* from the distribution and keep the samples. Once this supply is exhausted we compute a new table.

Alias table: Denote by

$$Q_w := \sum_t \alpha_t \frac{n_{tw} + \beta_w}{n_t + \bar{\beta}} \text{ and } q_w(t) := \frac{\alpha_t}{Q_w} \frac{n_{tw} + \beta_w}{n_t + \bar{\beta}}$$

the alias normalization and the associated probability distribution. Then we perform the following steps:

1. Generate the alias table A using q_w.
2. Draw k samples from q_w and store them in S_w.
3. Discard A and only retain Q_w and the array S_w.

Generating S_w and computing Q_w costs $O(k)$ time. In particular, storage of S_w requires at most $O(k \log_2 k)$ bits, thus it is much more compact than A. Note, however, that we need to store Q_w and $q_w(t)$.

Metropolis Hastings proposal: Denote by

$$P_{dw} := \sum_t n_{td}^{-di} \frac{n_{tw}^{-di} + \beta_w}{n_t^{-di} + \bar{\beta}} \text{ and } p_{dw}(t) := \frac{n_{td}^{-di}}{P_{dw}} \frac{n_{tw}^{-di} + \beta_w}{n_t^{-di} + \bar{\beta}}$$

the sparse document-dependent topic contribution. Computing it costs $O(k_d)$ time. This allows us to construct a proposal distribution

$$q(t) := \frac{P_{dw}}{P_{dw} + Q_w} p_{dw}(t) + \frac{Q_w}{P_{dw} + Q_w} q_w(t) \tag{10}$$

To perform an MH-step we then draw from $q(t)$ in $O(k_d)$ amortized time. The step from topic s to topic t is accepted with probability $\min(1, \pi)$ where

$$\pi = \frac{n_{td}^{-di} + \alpha_t}{n_{sd}^{-di} + \alpha_s} \cdot \frac{n_{tw}^{-di} + \beta_w}{n_{sw}^{-di} + \beta_w} \cdot \frac{n_s^{-di} + \bar{\beta}}{n_t^{-di} + \bar{\beta}} \cdot \frac{P_{dw} p_{dw}(s) + Q_w q_w(s)}{P_{dw} p_{dw}(t) + Q_w q_w(t)}$$

Note that the last fraction effectively removes the normalization in p_{dw} and q_w respectively, that is, we take ratios of unnormalized probabilities.

Complexity: To draw from q costs $O(k_d)$ time. This is so since computing P_{dw} has this time complexity, and so does the sampler for p_{dw}. Moreover, drawing from $q_w(t)$ is $O(1)$, hence it does not change the order of the algorithm. Note that repeated draws from q are only $O(1)$ since we can use the very same alias sampler also for draws from p_{dw}. Finally, evaluating π costs only $O(1)$ time. We have the following:

Lemma 3 *Drawing up to k steps in a Metropolis-Hastings proposal from $p(z_{di}|\text{rest})$ can be accomplished in $O(k_d)$ amortized time per sample and $O(k)$ space.*

4.2 Sampler for the Poisson Dirichlet Process

Following the same steps as above, the basic Poisson Dirichlet Process topic model can be decomposed by exploiting the sparsity of n_{dt}. The main difference to before is that we need to account for the auxiliary variable $r \in \{0, 1\}$ rather than just the topic indicator t. The alias table is:

$$q_w(t, r) := \begin{cases} \frac{\alpha_k}{b_t + m_{tw}} \frac{m_{tw} - s_{tw} + 1}{m_{tw} + 1} \frac{S_{s_{tw}, a_t}^{m_{tw}+1}}{S_{s_{tw}, a_t}^{m_{tw}}} & \text{if } r = 1 \\ \alpha_k \frac{b_t + a_t s_t}{b_t + m_t} \frac{s_{tw} + 1}{m_{tw} + 1} \frac{\beta + s_{tw}}{\bar{\beta} + s_t} \frac{S_{s_{tw}+1, a_t}^{m_{tw}+1}}{S_{s_{tw}, a_t}^{m_{tw}}} & \text{otherwise} \end{cases}$$

$$Q_w := \sum_{r,t} q_w(t, r)$$

Likewise, the sparse document-specific contribution is

$$p_{dw}(t, r) := n_{dt} \begin{cases} \frac{1}{b_t + m_t} \frac{m_{tw} - s_{tw} + 1}{m_{tw} + 1} \frac{S_{s_{tw}, a_t}^{m_{tw}+1}}{S_{s_{tw}, a_t}^{m_{tw}}} & \text{if } r = 1 \\ \frac{b_t + a_t s_{tw}}{b_t + m_{tw}} \frac{s_{tw} + 1}{m_{tw} + 1} \frac{\beta + s_{tw}}{\bar{\beta} + s_t} \frac{S_{s_{tw}+1, a_t}^{m_{tw}+1}}{S_{s_{tw}, a_t}^{m_{tw}}} & \text{otherwise} \end{cases}$$

$$P_{dw} := \sum_{r,t} p_{dw}(t, r)$$

As previously, computing $p_{dw}(t, r)$ only costs $O(k_d)$ time, which allows a proposal distribution very similar to the case in LDA to be constructed:

$$q(t, r) := \frac{P_{dw}}{P_{dw} + Q_w} p_{dw}(t, r) + \frac{Q_w}{P_{dw} + Q_w} q_w(t, r)$$

As before, we use a Metropolis-Hastings sampler, although this time for the state pair $(s, t) \to (s', t')$ and accept as before by using the ratio of current and stale probabilities (the latter given by q). As before in the context of LDA, the time complexity of this sampler is amortized $O(k_d)$.

4.3 Sampler for the HDP

Due to slight differences in the nature of the sparse term and the dense term, we demonstrate the efficacy of our approach for sparse language models here. That is, we show that whenever the document model is dense but the language model sparse, our strategy still applies. In other words, this sampler works at $O(k_w)$ cost which is beneficial for infrequent words.

For brevity, we only discuss the derivation for the two level HDP-LDA, given that the general multi-level HDP can be easily extended from the derivation. Recall (9). Now the alias table is now given by:

$$q_w(t, u) := p\left(z_{di} = t, u_{di} = u | \text{rest}\right) \left(\bar{\beta} + m_t\right) \beta_w$$

$$Q_w := \sum_{t,u} q_w(t, u)$$

and the exact term is given by

$$p_{dw}(t, u) := \gamma_w p(z_{di} = t, u_{di} = u | \text{rest}) (\bar{\gamma} + m_t) m_{tw}$$

$$P_{dw} := \sum_{t,u} p_{dw}(t, u)$$

As before, we engineer the proposal distribution to be a combination of stale and fresh counts. It is given by

$$q(t, u) := \frac{P_{dw}}{P_{dw} + Q_w} p_{dw}(t, u) + \frac{Q_w}{P_{dw} + Q_w} q_w(t, u)$$

Subsequently, the state transition $(t, u) \to (t', u')$ is accepted using straightforward Metropolis-Hastings acceptance ratios. We omitted the subscript $w_{di} = w$ for brevity. The same argument as above shows that the time complexity of our sampler for drawing from HDP-LDA is amortized $O(k_w)$.

5. EXPERIMENTS

To demonstrate empirically the performance of the alias method we implemented the aforementioned samplers in both their base forms that have $O(k)$ time complexity, as well as our alias variants which have amortized $O(k_d)$ time complexity. In addition to this, we implemented the SparseLDA [22] algorithm with the full set of features including the sorted list containing a compact encoding of n_{tw} and n_{dt}, as well as dynamic $O(1)$ update of bucket values. Beyond the standard implementation provided in MalletLDA

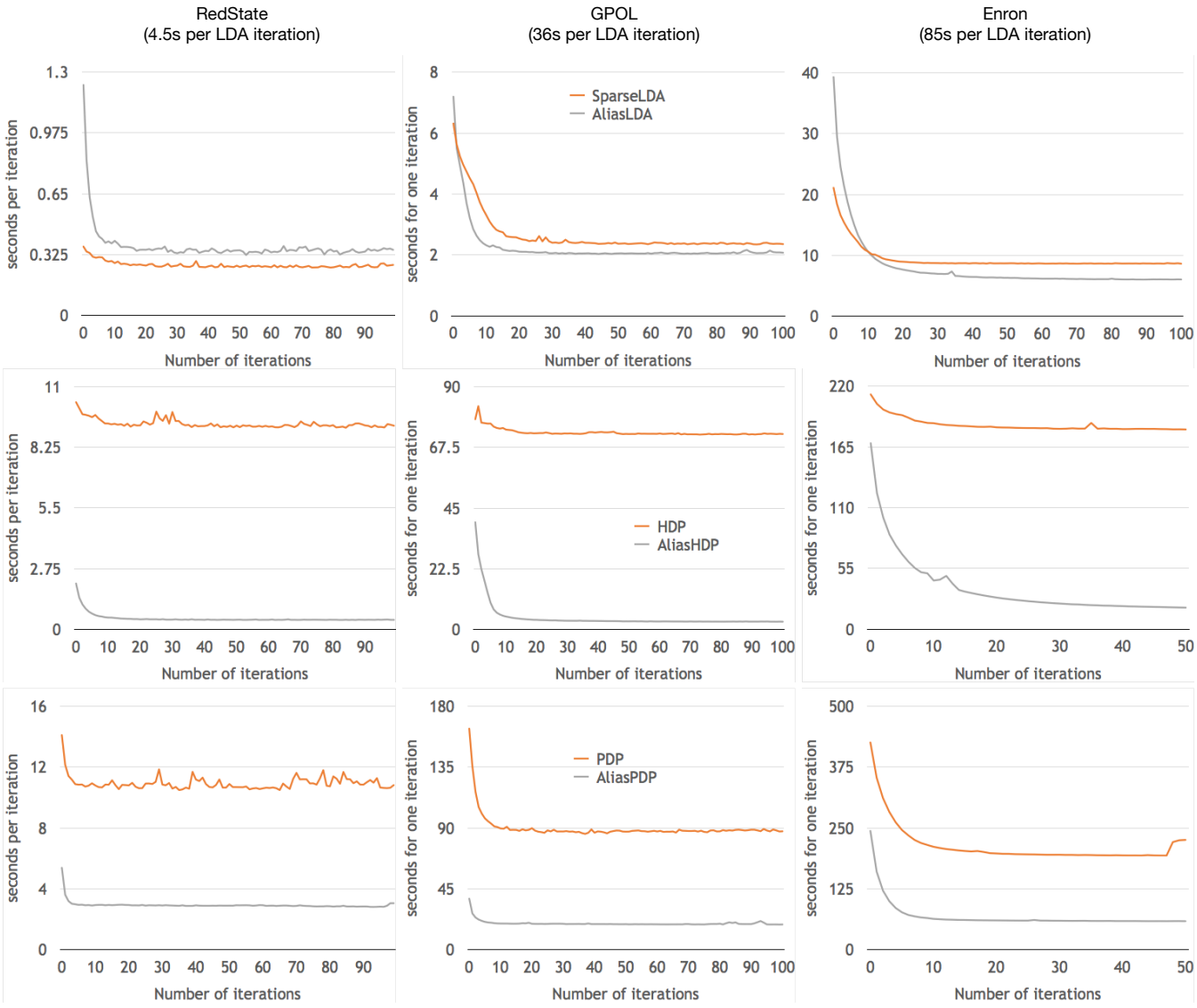


Figure 1: Runtime time comparison between LDA, HDP, PDP and their Alias sampled counterparts AliasLDA, AliasHDP and AliasPDP.

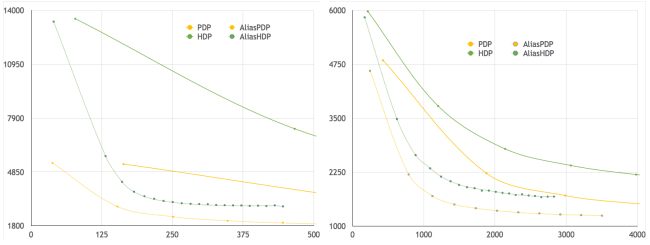


Figure 2: Perplexity as a function of runtime (in seconds) for PDP, AliasPDP, HDP, and AliasHDP on GPOL (left) and Enron (right).

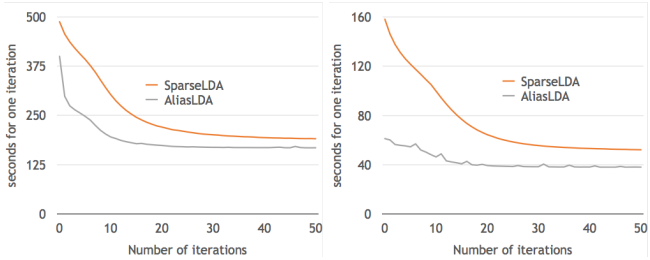


Figure 3: Runtimes of SparseLDA and AliasLDA on PubMedSmall (left) and NyTimes (right).

by [22], we made two major improvements: we accelerated the sorting algorithm for the compact list of encoded values to amortized $O(1)$; and we avoided hash maps which substantially improved the speed in general with small sacrifice of memory efficiency (we need an inverted list of the indices and an inverted list of the indices of the inverted lists).

In this section these implementations will be referred as **LDA** which is $O(k)$, **SparseLDA** which is $O(k_w + k_d)$,

AliasLDA which is $O(k_d)$, **PDP** at $O(k)$ [5], **AliasPDP** at $O(k_d)$, **HDP at** $O(k)$ [6], and **AliasHDP** at $O(k_w)$.

5.1 Environment and Datasets

All our implementations are written in `C++11` in a way that maximise runtime speed, compiled with `gcc 4.8` with `-O3` compiler optimisation in `amd64` architecture. All our experiments are conducted on a laptop with 12GB memory and an Intel i7-740QM processor with 1.73GHz clock rate, 4×256KB L2 Cache and 6MB L3 Cache. Furthermore, we only use one single sampling thread across all experiments. Therefore, only one CPU core is active throughout and only 256KB L2 cache is available. We further disabled Turbo Boost to ensure all experiment are run at exactly 1.73GHz clock rate. Ubuntu 13.10 64bit served as runtime.

We use 5 datasets with a variety in sizes, vocabulary length, and document lengths for evaluation, as shown in Table 1 . **RedState** dataset contains American political blogs crawled from redstate.com in year 2011. **GPOL** contains a subset of political news articles from Reuters RCV1 collection.[2] We also included the **Enron** Email Dataset,[3]. **NYTimes** contains articles published by New York Times between year 1987 and 2007. **PubMedSmall** is a subset of approximately 1% of the biomedical literature abstracts from PubMed. Stopwords are removed from all datasets. Furthermore, words occurring less than 10 times are removed from **NYTimes**, **Enron**, and **PubMedSmall**. **NYTimes**, **Enron**, and **PUBMED** datasets are available at [1].

5.2 Evaluation Metrics and Parameters

We evaluate the algorithms based on two metrics: the amount of time elapsed for one Gibbs sampling iteration and perplexity. The perplexity is evaluated after every 5 iterations, beginning with the first evaluation at the ending of the first Gibbs sampling iteration. We use the standard held-out method [10] to evaluate test perplexity, in which a small set of test documents originating from the same collection is set to query the model being trained. This produces an estimate of the document-topic mixtures $\tilde{\theta}_{dt}$ for each test document d. From there the perplexity is then evaluated as:

$$\pi(\mathbf{W}|\text{rest}) := \left[\sum_{d=1}^{D} N_d\right]^{-1} \sum_{d=1}^{D} \log p(w_d|\text{rest}) \text{ where}$$

$$p(w_d|\text{rest}) = \prod_{i=1}^{n_d} \sum_{t=1}^{k} p(w_i = w|z_{di} = t, \text{rest}) p(z_{di} = t|\text{rest})$$

Here we obtain the estimate of $p(w_i = w|z_{di} = t, \text{rest})$ from the model being trained. To avoid effects due to variation in the number of topics, we hardcoded $k = 1024$ for all experiments except one (GPOL) where we vary k and observe the effect on speed per iteration. We use fixed values for hyperparameters in all our models, setting $\alpha = \beta = 0.1$ for LDA, $a = 0.1$, $b = 10$, and $\gamma = 0.1$ for the PDP, and $b_0 = b_1 = 10$, $\gamma = 0.1$ for the HDP. For alias implementations, we fix the number of Metropolis-Hasting sampling steps at 2, as we observed a satisfactory acceptance rate (over 90%) at these settings. Only a negligible improvement in perplexity was observed by raising this value. Furthermore, we did not observe degraded topic quality even when Metropolis-Hasting sampling step was reduced to $n = 1$, and in all our experiments the perplexity almost perfectly converges at the same pace (i.e. along number of iterations) with the same algorithm without applying alias method (albeit with much less time per iteration).

[2]Reuters Vol. 1, English language, 1996-08-20 to 1997-08-19
[3]Source: www.cs.cmu.edu/~enron

Dataset	V	L	D	T	L/V	L/D
RedState	12,272	321,699	2,045	231	26.21	157
GPOL	73,444	2,638,750	14,377	1,596	35.9	183
Enron	28,099	6,125,138	36,999	2,860	218	165
PubMedSmall	106,797	35,980,539	546,665	2,002	337	66
NYTimes	101,636	98,607,383	297,253	2,497	970	331

Table 1: Datasets and their statistics. V: vocabulary size; L: total number of training tokens, D: number of training documents; T: number of test documents. L/V is the average number occurrences of a word. L/D is the average document length.

5.3 Performance Summary

Figure 6 shows the overall performance of perplexity as a function of time elapsed when comparing SparseLDA vs AliasLDA on the four larger datasets. When k is fixed to 1024, substantial performance in perplexity over running time on all problems with the exception of the Enron dataset, most likely due to its uniquely small vocabulary size. The gap in performance is increased as the datasets become larger and more realistic in size. The gain in performance is noted in particular when the average document length is smaller since our sampler scales with $O(k_d)$ which is likely to be smaller for short documents.

Figure 2 gives the comparison between PDP, HDP and their aliased variants on GPOL and Enron. By the time AliasPDP and AliasHDP are converged, the straightforward sampler are still at their first few iterations.

5.4 Performance as a Function of Iterations

In the following we evaluate the performance in two separate parts: perplexity as a function of iterations and runtime vs. iterations. We first establish that the acceleration comes at no cost of degraded topic quality, as shown in Figure 6. The convergence speed and converged perplexity of AliasLDA, AliasPDP, and AliasHDP almost perfectly match the non-alias counterparts. This further shows that our choice of relatively small number of Metropolis-Hasting steps (2 per sample) is adequate.

The improved performance in running time of our alias implementations can be seen in all phases of sampling when compared to non-alias standard implementations (LDA, PDP, HDP). When compared to SparseLDA (Figure 3), the performance gain is salient during all phases on larger datasets (except for the early stage in Enron dataset), and the performance is very close on small datasets (0.35s per iteration on AliasLDA vs. 0.25s per iteration on SparseLDA). As the size of the data grows AliasLDA outperforms SparseLDA without degrading topic quality, reducing the amount of time for each Gibbs iteration on NYTimes corpus by around 12% to 38% overall, on Enron corpus by around 30% after 30 iterations, and on PubMedSmall corpus by 27%-60% throughout the first 50 iterations. Compared to SparseLDA, the time required for each Gibbs iteration with AliasLDA grows at a much slower rate, and the benefits of reduced sampling complexity is particularly clear when the average length of each document is small.

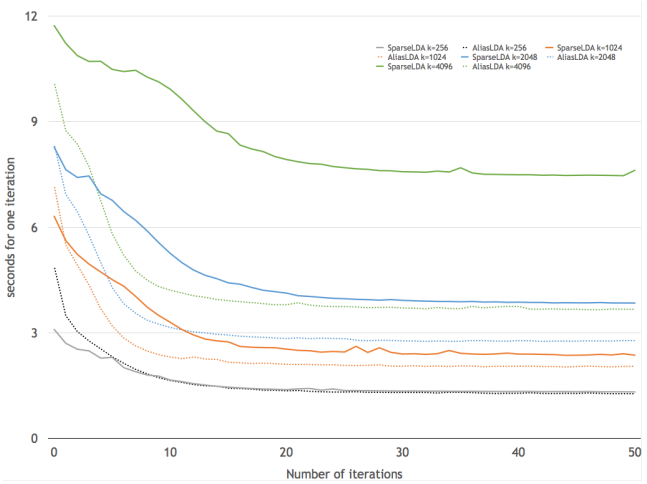


Figure 4: Comparison of SparseLDA and AliasLDA on GPOL when varying the number of topics for $k \in \{256, 1024, 2048, 4096\}$.

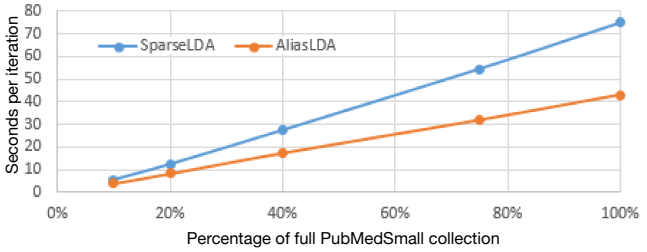


Figure 5: Average runtime per iteration when compared on $\{10\%, 20\%, 40\%, 75\%, 100\%\}$ of the PubMedSmall dataset for SparseLDA and AliasLDA.

The gap in performance is especially large for more sophisticated language modelsl such as PDP and HDP. The running time for each Gibbs iteration is reduced by 60% to 80% for PDP, and 80% to 95% for HDP, an order of magnitude on improvement.

5.5 Varying the number of topics

When the number of topics k increases, the running time for an iteration of AliasLDA increases at a much lower rate than SparseLDA, as seen from Figure 4 on dataset GPOL since k_d is almost constant. Even though the gap between SparseLDA and AliasLDA may seem insignificant at $k = 1024$, it becomes very pronounced at $k = 2048$ (45% improvement) and at $k = 4096$ (over 100%) This confirms the observation above that shorter documents benefits more from AliasLDA in the sense that the average documents length L/D relative to the number of topics k becomes "shorter" as k increases. This yields a more sparse n_{dt} and lower k_d for a document d on average.

5.6 Varying the corpus size

Figure 5 demonstrates how the gap in running time speed scales with growing number of documents in the same domain. We measure the average runtime for the first 50 Gibbs iterations on 10%, 20%, 40%, 75%, and 100% of PubMedSmall dataset. The speedup ratio for each subset is at 31%, 34%, 37%, 41%, 43% respectively. In other words, it increases with the amount of data, which conforms our intuition that adding new documents increases the density of n_{tw}, thus slowing down the sparse sampler much more than the alias sampler, since the latter only depends on k_d rather than $k_d + k_w$.

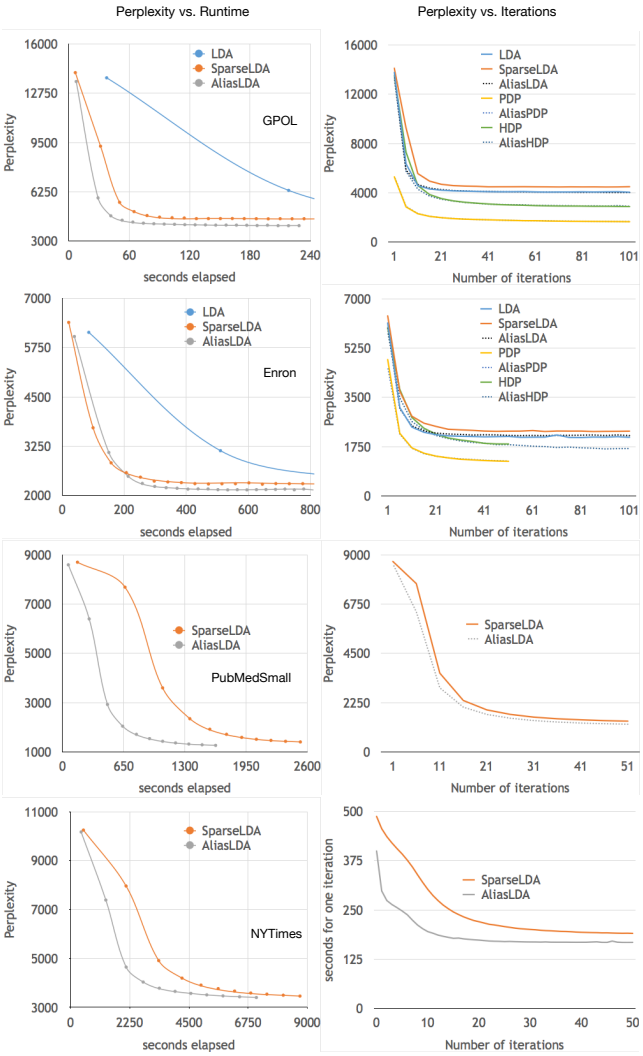


Figure 6: Perplexity as a function of runtime (left) and number of iterations (right) for LDA, SparseLDA, and LDA, PDP and HDP, both with and without using the Alias method. We see considerable acceleration at unchanged perplexity.

6. CONCLUSION

In this paper, we described an approach that effectively reduces sampling complexity of topic models from $O(k)$ to $O(k_d)$ in general, and from $O(k_w+k_d)$ (SparseLDA) to $O(k_d)$ (AliasLDA) for LDA topic model. Empirically, we showed that our approach scales better than existing state-of-the-art method when the number of topics and the number of documents become large. This enables many large scale applications, and many existing applications which require a

scalable distributed approach. In many industrial applications where the number of tokens easily reaches billions, these properties are crucial and often desirable in designing a scalable and responsive service. We also demonstrated an order of magnitude improvement when our approach is applied to complex models such as PDP and HDP. With an order of magnitude gain in speed, PDP and HDP may become much more appealing to many applications for their superior convergence performance, and more sophisticated representation of topic distributions and language models.

For $k = 1024$ topics the number of tokens processed per second in our implementation is beyond 1 million for all datasets except one (NYTimes), of which contains substantially more lengthy documents. This is substantially faster than many known implementations when measured in number of tokens processed per computing second per core, such as YahooLDA [18], and GraphLab, given that we only utilise a single thread on a single laptop CPU core.

Acknowledgments: This work was supported in part by a resource grant from amazon.com, a Faculty Research Grant from Google, and Intel.

7. REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] D. Blei, T. Griffiths, and M. Jordan. The nested chinese restaurant process and Bayesian nonparametric inference of topic hierarchies. *Journal of the ACM*, 57(2):1–30, 2010.

[3] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, Jan. 2003.

[4] W. Buntine and M. Hutter. A bayesian review of the poisson-dirichlet process, 2010.

[5] C. Chen, W. Buntine, N. Ding, L. Xie, and L. Du. Differential topic models. In *IEEE Pattern Analysis and Machine Intelligence*, 2014.

[6] C. Chen, L. Du, and W. Buntine. Sampling table configurations for the hierarchical poisson-dirichlet process. In *ECML*, pages 296–311, 2011.

[7] J. Geweke and H. Tanizaki. Bayesian estimation of state-space model using the metropolis-hastings algorithm within gibbs sampling. *Computational Statistics and Data Analysis*, 37(2):151–170, 2001.

[8] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. 1995.

[9] T. Griffiths and M. Steyvers. Finding scientific topics. *PNAS*, 101:5228–5235, 2004.

[10] G. Heinrich. Parameter estimation for text analysis. Technical report, Fraunhofer IGD, 2004.

[11] M. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. In *ICML*, 2012.

[12] W. Li, D. Blei, and A. McCallum. Nonparametric bayes pachinko allocation. In *UAI*, 2007.

[13] G. Marsaglia, W. W. Tsang, and J. Wang. Fast generation of discrete random variables. *Journal of Statistical Software*, 11(3):1–8, 2004.

[14] R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. University of Toronto, Technical Report 9815, 1998.

[15] J. Petterson, A. Smola, T. Caetano, W. Buntine, and S. Narayanamurthy. Word features for latent dirichlet allocation. In *NIPS*, pages 1921–1929, 2010.

[16] J. Pitman and M. Yor. The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *A. of Probability*, 25(2):855–900, 1997.

[17] I. Sato and H. Nakagawa. Topic models with power-law using Pitman-Yor process. In *KDD*, pages 673–682. ACM, 2010.

[18] A. J. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *PVLDB*, 2010.

[19] Y. Teh, M. Jordan, M. Beal, and D. Blei. Hierarchical dirichlet processes. *JASA*, 101(576):1566–1581, 2006.

[20] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM TOMS*, 3(3):253–256, 1977.

[21] C. Wang, J. Paisley, and D. M. Blei. Online variational inference for the hierarchical Dirichlet process. In *Conference on Artificial Intelligence and Statistics*, 2011.

[22] L. Yao, D. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In *KDD'09*, 2009.