# A Smart-Dumb/Dumb-Smart Algorithm for Efficient Split-Merge MCMC

**Wei Wang**
LIP6
Université Pierre et Marie Curie, 75005 Paris
benwei.wang@outlook.com

**Stuart Russell**
Computer Science Division
University of California, Berkeley, CA 94720
russell@cs.berkeley.edu

## Abstract

Split-merge moves are a standard component of MCMC algorithms for tasks such as multi-target tracking and fitting mixture models with unknown numbers of components. Achieving rapid mixing for split-merge MCMC has been notoriously difficult, and state-of-the-art methods do not scale well. We explore the reasons for this and propose a new split-merge kernel consisting of two sub-kernels: one combines a "smart" split move that proposes plausible splits of heterogeneous clusters with a "dumb" merge move that proposes merging random pairs of clusters; the other combines a dumb split move with a smart merge move. We show that the resulting smart-dumb/dumb-smart (SDDS) algorithm outperforms previous methods. Experiments with entity-mention models and Dirichlet process mixture models demonstrate much faster convergence and better scaling to large data sets.

## 1 INTRODUCTION

Markov Chain Monte Carlo (MCMC) algorithms have become a central pillar of statistical inference and machine learning. MCMC algorithms repeatedly apply a stochastic *kernel* transformation to an initial state, generating a random walk through the sample space whose stationary distribution matches a desired target distribution. Within the general Metropolis-Hastings (MH) family of MCMC methods, which includes many specific algorithms that have proven useful in practice, the kernel is built from a *proposal* step followed by a stochastic *acceptance* step whose probability is determined by the chosen proposal distribution. One key to efficient MH inference, then, is proposal design.[1]

This paper focuses on MH proposal design for *split–merge* moves. Split and merge moves, which form a complementary pair comprising a kernel, are useful for problems where an MCMC state can be thought of as consisting of a number of components or clusters, each of which is responsible for some subset of observations. The canonical example is the family of *mixture models*: a split move in such a model converts one mixture component into two, dividing the observations of the original component between the two new components; a merge move combines two components and their observations into a single component [Dahl, 2003; Pasula *et al.*, 2003; Jain and Neal, 2004]. Split-merge moves are also common in multitarget tracking, where a "component" is a single track joining together observations of an object over multiple time steps [Pasula *et al.*, 1999; Khan *et al.*, 2005]. The state of the art for split-merge MCMC in mixture models is considered to be the *restricted Gibbs split-merge* (RGSM) algorithm of Jain and Neal [2004].

The general idea followed in most MH proposal designs is to make the proposal "smart" by preferentially proposing states with higher probability according to the target distribution. This tends to give higher acceptance probabilities. The Gibbs sampler [Geman and Geman, 1984] is the quintessential smart proposal: by proposing values for some subset of variables exactly in proportion to their probability, Gibbs sampling has an acceptance probability of 1. In the context of split-merge moves, a smart split would be one that favors splitting a heterogeneous cluster to produce two more homogeneous ones, and a smart merge would favor merging similar clusters to ensure a homogeneous result. As our analysis and experiments show, however, combining smart split and merge moves does not lead to high acceptance probabilities and rapid convergence. The reason is the asymmetry between subspaces with $K$ and $K+1$ components: there are far more states in the latter than the former, whereas in the case of Gibbs sampling the source and target subspaces are identical.

[1] Other approaches include parallelization [Chang and Fisher, 2013; Williamson *et al.*, 2013] and taking advantage of symmetry [Niepert and Domingos, 2014]. The work reported in this paper can easily be combined with these approaches.

Our proposed solution, the smart-dumb/dumb-smart (SDDS) algorithm, combines two kernels in parallel: one has a smart split move and a "dumb" merge move that proposes merging random pairs of clusters; the other combines a dumb split move with a smart merge move. We show that the resulting algorithm performs well in practice, maintaining high acceptance rates and converging reasonably quickly even for large data sets with many clusters, where RGSM and other algorithms fail. To our knowledge, the closest relative of SDDS is a parallel-computation MCMC algorithm due to Chang and Fisher [2013], who mention the use of a random split move as the complement of a merge move in one part of a rather complex algorithm. A second contribution of our paper is a fast, exact method for sampling a split move from the posterior over all possible splits of a given component, i.e., an efficient block-Gibbs proposal for splits.

The paper begins (Section 2) with background material on MCMC. Section 3 describes, for expository purposes, the *entity/mention model* (EMM), a very simple Bayesian mixture model with observations that are discrete tokens, and examines split-merge MCMC in the context of the EMM. Section 4 describes the SDDS algorithm in detail. Finally, Section 5 evaluates SDDS in comparison to other approaches, both on EMM data and on data from a Dirichlet process mixture model (DPMM).

## 2 MCMC METHODS

Here we provide a brief review of the relevant aspects of Metropolis–Hastings MCMC. Let $\mathcal{X}$ be a sample space (a set of possible worlds); a sample point $\mathbf{x} \in \mathcal{X}$ will be called a *state*. Let $\pi(\cdot)$ be a *target distribution* of interest, such as the posterior distribution on $\mathcal{X}$ given some evidence. The goal is to generate samples from $\pi$, or something close to it, so as to answer queries. A standard MCMC algorithm constructs a Markov chain from a *transition kernel* $P(\mathbf{x}'|\mathbf{x})$, such that the unique stationary distribution of the chain is $\pi$; of primary concern is the rate of convergence of the Markov chain to its stationary distribution.

The Metropolis–Hastings (MH) algorithm [Metropolis *et al.*, 1953; Hastings, 1970] is a general template for building transition kernels with the desired property. Each transition is built from two steps: first, a new state $\mathbf{x}'$ is proposed from a *proposal distribution* $q(\mathbf{x}'|\mathbf{x})$, then the new state is accepted with a probability given by

$$\alpha(\mathbf{x}'|\mathbf{x}) = min\{1, \frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} \frac{q(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})}\} \,. \tag{1}$$

(If the proposal is not accepted, the new state is the same as the current state.) The ratio appearing in this expression is called the *MH ratio*; we have written it as the product of the *state ratio* $\pi(\mathbf{x}')/\pi(\mathbf{x})$ and the *proposal ratio* $q(\mathbf{x}|\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})$. The only "free parameter" in designing an MH algorithm is the proposal $q(\cdot|\cdot)$, and it is this that determines the rate of convergence.

Gibbs sampling can be understood as a special case of MH [Gelman, 1992]. In its simplest form, it chooses a variable $X_i$ uniformly at random from the set of $n$ variables whose values define the state $\mathbf{x}$ and proposes a value $x_i'$ from the distribution $\pi(X_i|\mathbf{x}_{-i})$, where $\mathbf{x}_{-i}$ denotes the current values for all variables other than $X_i$. Because this proposal distribution is proportional to the state probability, the proposal ratio for Gibbs is exactly the inverse of the state ratio. For the case where $x_i$ and $x_i'$ coincide, both ratios are 1; when they differ, we have

$$\begin{aligned} \frac{q(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})} &= \frac{q(x_i, \mathbf{x}_{-i}|x_i', \mathbf{x}_{-i})}{q(x_i', \mathbf{x}_{-i}|x_i, \mathbf{x}_{-i})} = \frac{\frac{1}{n}\pi(x_i|\mathbf{x}_{-i})}{\frac{1}{n}\pi(x_i'|\mathbf{x}_{-i})} \\ &= \frac{\pi(x_i, \mathbf{x}_{-i})\pi(\mathbf{x}_{-i})}{\pi(x_i', \mathbf{x}_{-i})\pi(\mathbf{x}_{-i})} = \frac{\pi(\mathbf{x})}{\pi(\mathbf{x}')} \,. \end{aligned} \tag{2}$$

Hence the acceptance probability in (1) is therefore exactly 1 for Gibbs sampling.

Having a high acceptance probability—or at least, one bounded away from zero—is a necessary but not sufficient condition for rapid mixing in MH. Moves can be accepted with high probability but if those moves fail to lead the chain from one local maximum in $\pi$ to another, overall mixing may still be slow. Thus, good proposal design is concerned with both acceptance probabilities and the ability to traverse the state space without getting stuck in local maxima.

## 3 THE ENTITY/MENTION MODEL

The entity/mention model or EMM is a very simple form of mixture model, defined here for the purposes of exposition. The EMM posits a certain (unknown) number of entities that are referred to by some set of mentions. For example, there is a person who may variously be referred to as "Barack Obama", "the President", "POTUS", and so on. Given a collection of mentions of various entities—for example, in newspaper text—the task is to figure out how many entities exist, which mentions refer to which entities, and thence the ways in which any given entity may be mentioned. In its simplest form, the EMM assumes that each mention is a token with no internal structure, drawn from a fixed, known set of tokens. This renders the model less interesting than the models used in NLP research, but has the advantage of simplifying the analysis.

### 3.1 The EMM probability model

We assume $N$ mentions and $L$ possible tokens. An EMM model is composed from the following variables and conditional distributions:

- $K$, the number of entities, drawn from a prior $P(K)$.

- For each entity $k$, a *dictionary* $\boldsymbol{\theta}_k$, i.e., a categorical distribution over $L$ tokens, drawn from a $Dirichlet(\boldsymbol{\alpha}_k)$. The set of dictionaries $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$ is represented by $\boldsymbol{\Theta}$.
- For each mention $m_n$, the entity $S_n$ for that mention is drawn u.a.r. from the set of $K$ entities, and the token for that entity is drawn from $\boldsymbol{\theta}_{S_n}$. The (unknown) entities $\{S_1, \ldots, S_N\}$ are represented by $\mathbf{S}$ and the observed mentions $\{m_1, \ldots, m_N\}$ by $\mathbf{m}$.

The EMM resembles a topic model for a single document with an unknown number of topics (entities).

In the experiments described below, we use a broad prior for $K$, namely a discretized log-normal distribution with the location parameter $\mu$ and the scale parameter $\sigma$ on a logarithmic scale:

$$P(K) = \frac{1}{C}\frac{1}{K\sigma\sqrt{2\pi}}e^{-(\log K-\mu)^2/2\sigma^2}$$

where $C$ is an additional normalization factor arising from discretizing the distribution.

Because the entity for any given mention is assumed to be chosen u.a.r. from the available entities, we have

$$P(\mathbf{S}|K) = \left(\frac{1}{K}\right)^N .$$

Rather than sample the dictionaries, we will integrate them out exactly, taking advantage of properties of the Dirichlet. In particular, we have

$$\int_{\boldsymbol{\Theta}} P(\mathbf{m}, \boldsymbol{\Theta}|K, \mathbf{S}) = \prod_{k\in\{1:K\}} \frac{B(\boldsymbol{\alpha}_k + \mathbf{n}_k)}{B(\boldsymbol{\alpha}_k)}$$

where $B(\cdot)$ is the Beta function and $\boldsymbol{\alpha}_k$ and $\mathbf{n}_k$ are both vectors of size $L$, representing respectively the Dirichlet prior counts and the observed counts of each token in the dictionary.

### 3.2 Gibbs sampling for the EMM

Basic Gibbs sampling samples one variable at a time, which means, in our entity/mention model, sampling a new entity assignment $S_n$ for some mention $m_n$, conditioned on all other current assignments $\mathbf{S}_{-n}$ of mentions to entities. The initial assignment is chosen at random, then the Gibbs sampler is repeated for I iterations, each cycling through all the mentions; see Algorithm 1.

The probability $P(S_n|K, \mathbf{S}_{-n}, \mathbf{m})$ is calculated by:

$$P(S_n = k|K, \mathbf{S}_{-n}, \mathbf{m}) \propto \frac{\alpha_{k,l} + n_{k,l}}{\sum_{l'\in L}(\alpha_{k,l'} + n_{k,l'})}$$

where $\alpha_{k,l}$ and $n_{k,l}$ are respectively the prior counts for the token $l$ (the mention $m_n = l$) and observed counts for token $l$ assigned to $E_k$. It is divided by the sum of the prior and observed counts for all the L tokens.

**Algorithm 1** Gibbs sampling for an entity/mention model

```
procedure GIBBS SAMPLING
    for n=1 to N do
        Sample S_n u.a.r. from {1, ..., K}
    end for
    for i=1 to I do
        for n=1 to N do
            Sample S_n from P(S_n|K, S_{-n}, m)
        end for
    end for
end procedure
```

Each step of the Gibbs sampler is relatively easy to compute, but of course the algorithm cannot change the number of entities; moreover, it tends to get stuck on local maxima because of the local nature of the changes [Celeux *et al.*, 2000]. Despite these drawbacks, Gibbs steps are an important element used in association with split-merge steps in order to optimize the allocation of mentions to entities.

### 3.3 Split–merge MCMC for the EMM

One way to explore states with different numbers of entities is to use *birth* and *death* moves. A birth moves creates a new entity with no mentions, while a death move kills off an entity that has no mentions. While such moves, combined with Gibbs moves, do connect the entire state space, they lead to very slow mixing because death moves can only occur when Gibbs moves have removed all the mentions from an entity, which is astronomically unlikely when $N$ is much larger than $K$—unless the entity being killed is one that was just born.

Split and merge moves simultaneously change the number of entities and change the assignments of multiple mentions in one go. The simplest approach is to pick an entity at random and split it into two, randomly assigning the mentions to the two new entities; the merge move operates in reverse by picking two entities and merging into one, along with their mentions. A naive implementation of this idea often fails to work, because random splits often yield a state with a very low probability, preventing acceptance. Pasula *et al.* [2003] suggested a *random mixing* procedure that chooses two entities and randomly assigns each of their mentions to one of two new entities. A split occurs when the two chosen entities happen to coincide, and a merge happens when one of the new entities receives no mentions and is discarded. The approach was effective for medium-sized data sets in their experiments (300-400 mentions, 60-80 entities) but fails when each entity has many mentions: merges become exponentially unlikely to be proposed.

Jain and Neal [2004] proposed a Restricted Gibbs Split–

Merge (RGSM) algorithm to generate splits that are consistent with data. Two random elements are chosen in the beginning. A split is proposed if these two elements belong to the same component and otherwise a merge is proposed. To split the component $c_k$, RGSM algorithm first assigns the elements randomly into two new components $c_1^{launch}$ and $c_2^{launch}$ as the launch state. Restricted Gibbs is then applied for $t$ times inside the launch state, re-assigning elements to one of the components. The modified launch state after $t$ Restricted Gibbs steps is used for generating the split. The resulting split reflects the data to some extent and tends to have a higher likelihood. However, these intermediate Restricted Gibss steps are rather computationally expensive, especially for large data sets. Dahl [2003] proposed an allocation procedure, which works by assigning elements sequentially to two components. It starts with creating two new components $c_1$ and $c_2$ with two random elements. The remaining elements are sequentially allocated to either $c_1$ or $c_2$ using the Restricted Gibbs sampler conditioned on those previously assigned elements. This procedure is more efficient than preparing the launch state in RGSM.

Split–merge has been applied to different models such as the Beta Process Hidden Markov Model [Hughes *et al.*, 2012] and the Hierarchical Dirichlet Process [Wang and Blei, 2012; Rana *et al.*, 2013]; on the other hand, the split–merge method itself has not been improved since RGSM was first proposed in 2004. In the next section, we examine the interaction of the MH algorithm with split–merge moves and propose a new combination of moves that seems to work better.

# 4 SMART AND DUMB PROPOSALS

In general, smart proposals that lead to high-probability states are preferred, as they lead to faster convergence of MCMC. What Jain and Neal [2004] did for RGSM is to avoid the low-probability states generated by random splits. As we mentioned in Section 1, "smart" proposals propose states with higher probability according to the target distribution. The Gibbs sampler is smart in this sense, because its proposal distribution is proportional to the state probability and hence the MH acceptance probability is always 1 (Eq. 2). Consequently, we may instinctively conclude that the convergence efficiency might be significantly improved if we concentrate on the design of smart proposals. However, this is not true for MH in general. The MH ratio in the case of a smart merge proposal will be analyzed as an example.

Let $q(\mathbf{x}'|\mathbf{x})$ and $q(\mathbf{x}|\mathbf{x}')$ be respectively a smart merge proposal and a smart split proposal. Each first picks a subset of the variables to merge (or split) with probability $P_m$ (or $P_s$). It then proposes a particular merge (or split) according to the target distribution $f_m$ (or $f_s$), where $f_m$ and $f_s$ are proportional to state probabilities:

$$f_m = \frac{\pi(\mathbf{x}')}{\sum_{\omega \in W(\mathbf{x})} \pi(\omega)},\ f_s = \frac{\pi(\mathbf{x})}{\sum_{\omega \in W(\mathbf{x}')} \pi(\omega)}, \qquad (3)$$

where $W(\mathbf{x})$ and $W(\mathbf{x}')$ are respectively the set of states for all possible merges and the set of states for all possible splits given $P_s$ and $P_m$.

The MH ratio is then given by

$$\begin{aligned} \frac{q(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})}\frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} &= \frac{P_s}{P_m}\frac{\frac{\pi(\mathbf{x})}{\sum_{\omega \in W(\mathbf{x}')} \pi(\omega)}}{\frac{\pi(\mathbf{x}')}{\sum_{\omega \in W(\mathbf{x})} \pi(\omega)}}\frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} \\ &= \frac{P_s}{P_m}\frac{\sum_{\omega \in W(\mathbf{x})} \pi(\omega)}{\sum_{\omega \in W(\mathbf{x}')} \pi(\omega)} \qquad (4) \end{aligned}$$

When both split and merge are smart, the ratio $\frac{P_s}{P_m}$ will be very low due to a quite small $P_s$. It is because that the smart split would not give a big probability mass to the part a smart merge prefers to merge (detailed examples given in the next subsection). The second part in Formula 4, namely space ratio, is important in split-merge case because of the space asymmetry. $\sum_{\omega \in W(\mathbf{x})}$ on the top is just $\pi(\mathbf{x})$ since when we have chosen two entities to merge ($P_m$), there is only one merge possibility. However, $\sum_{\omega \in W(\mathbf{x}')}$ contain $2^n$ possible splits (n is the number of mentions assigned to the picked entity).

The $\frac{P_s}{P_m}$ ratio gives the first idea about why smart proposals have conflicts with inverse smart proposals. In case of space asymmetry, a smart–smart proposal suffers also from the space ratio in addition to the $\frac{P_s}{P_m}$ ratio.

## 4.1 Why smart proposals do not work by themselves: A simple example

Let's take a concrete example of split and merge to illustrate the problem stated above for smart proposals. Assuming that there are three entities in state $\mathbf{x}$ as below:

$$\mathbf{x} : \{\mathbf{E_1} : \{\text{A A B B}\};\ \mathbf{E_2} : \{\text{C C}\};\ \mathbf{E_3} : \{\text{C C}\}\},$$

a desired split would be splitting $\mathbf{E_1}$ into two entities as follows:

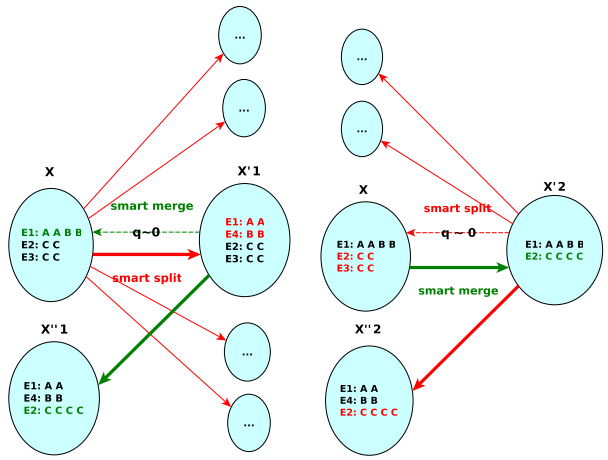
$$\mathbf{x}'_1 : \{\mathbf{E_1} : \{\text{A A}\};\ \mathbf{E_4} : \{\text{B B}\};\ \mathbf{E_2} : \{\text{C C}\};\ \mathbf{E_3} : \{\text{C C}\}\}.$$

A smart split proposal distribution should propose the state $\mathbf{x}'_1$ with a quite high probability as illustrated in the left part of Figure 1, where the width of the arrow line indicates the probability value. However, a smart merge proposal, starting from state $\mathbf{x}'_1$, would rather have a high probability $q(\mathbf{x}''_1|\mathbf{x}'_1)$ for proposing the state $\mathbf{x}''_1$:

$$\mathbf{x}''_1 : \{\mathbf{E_1} : \{\text{A A}\};\ \mathbf{E_4} : \{\text{B B}\};\ \mathbf{E_2} : \{\text{C C C C}\}\}.$$

From the point of view of a smart merge, inverting the smart split's preferred move $q(\mathbf{x}|\mathbf{x}'_1)$ is highly unlikely, as

represented by the dashed arrow in the figure. Therefore, considering the high value of $q(\mathbf{x}'_1|\mathbf{x})$, the proposal ratio becomes extremely low, which leads to a very low acceptance rate for smart split proposals.




(a) smart split (b) smart merge

Figure 1: Conflicts between smart split and smart merge (red lines for splits and green lines for merges; thick/dashed lines for moves preferred/dispreferred by smart proposals.)

It is a similar situation for smart merge proposals, as illustrated in the right part of Figure 1. A smart merge proposal will give a high probability to generate the state $\mathbf{x}'_2$:

$$\mathbf{x}'_2 : \{\mathbf{E_1} : \{\text{A A B B}\};\ \mathbf{E_2} : \{\text{C C C C}\}\}.$$

On the other hand, the smart split from the state $\mathbf{x}'_2$ will be more likely to generate a state such as:

$$\mathbf{x}''_2 : \{\mathbf{E_1} : \{\text{A A}\};\ \mathbf{E_3} : \{\text{B, B}\};\ \mathbf{E_2} : \{\text{C C C C}\}\},$$

leaving only a tiny probability to propose the state $\mathbf{x}'_2$ to go back to $\mathbf{x}$. The same phenomenon of a low acceptance rate will be engendered.

For the particular case of split and merge, a new entity created by the split proposal changes the parameters of the space, which means, for one split and its inverse merge, there are much more possibilities for splits than those for merges. This neighborhood issue makes the smart merge proposal even harder to be accepted as shown in Formula 4.

### 4.2 Coupling with smart and dumb proposals

Smart proposals are not effective in this case because the entity which we choose to split does not correspond to the entities that we prefer to merge in the reverse direction. However, if we want to distribute a higher probability to the reverse move, $q(\mathbf{x}|\mathbf{x}'_1)$ for instance, it can be treated as a dumb proposal rather than a smart one. "Dumb" proposals can be considered as distributions that give uniform probability mass over all possible moves.

An important property of MCMC methods is that *detailed balance* can be guaranteed when several different proposals are adopted on condition that each proposal satisfies the Metropolis-Hastings algorithm [Tierney, 1994]. Therefore, there is a solution to combine smart and dumb proposals, namely the *Smart-Dumb Dumb-Smart* proposals (SDDS), as illustrated in Figure 2.

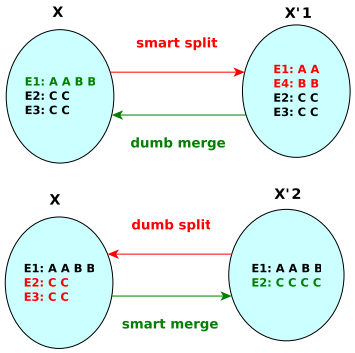



Figure 2: Smart-Dumb Dumb-Smart design for proposals

As a consequence, there are two separate pairs of proposal distributions. For either of them, the dumb proposal gives a uniform distribution over all possible moves. The existence of dumb proposals helps the acceptance of smart proposals. In the context of these two pairs, both smart split and smart merge can produce higher acceptance rates and faster mixing.

### 4.3 An SDDS split–merge algorithm

Inside the SDDS algorithm, we want to propose high-probability states with smart splits and smart merges and to do so efficiently. The algorithm for the smart split proposal with dumb merge and the one for smart merge proposal with dumb split are respectively described in Algorithm 2 and Algorithm 3.

**Smart split/dumb merge proposal** Algorithm 2 begins by choosing randomly between a smart split proposal and a dumb merge proposal. If a smart split is picked, it will first choose one entity $E_k$ based on a function $f_{split}(E_k)$. The function is inversely proportional to the likelihood of the mentions $\mathbf{m}_{E_k}$ associated with this entity $E_k$, which implies that the proposal tends to choose large and mixed entities. The likelihood is given by $B(\boldsymbol{\alpha}_k + \mathbf{n}_k)/B(\boldsymbol{\alpha}_k)$ where $\boldsymbol{\alpha}_k$ and $\mathbf{n}_k$ are respectively the vectors for the Dirichlet prior counts and the observed counts of each token in $\mathbf{m}_{E_k}$.

Once the entity $E_k$ chosen, the smart split procedure will allocate sequentially each mention to one of two newly created entities $E'_1$ and $E'_2$, according to the likelihood of the previously assigned mentions. Given the entity $E_k$ : $\{AABBCC\}$ for example, the procedure is illustrated in

**Algorithm 2** Smart Split and Dumb Merge proposal

**procedure** SMART SPLIT DUMB MERGE
  choose a move type: $type \sim (split, merge)$
  **if** type==split **then** ▷ smart split
    choose one entity to split

$$E_k \sim f_{split}(E_k)$$

$$f_{split}(E_k) \propto \frac{1}{P(\mathbf{m}_{E_k}|E_k)}$$

    create two new empty entities $E'_1$ and $E'_2$
    assign each mention in $E_k$ sequentially to $E'_1$ or $E'_2$ according to the likelihood of previously assigned mentions
  **else** ▷ dumb merge
    choose one entity uniformly from K entities
    choose another entity uniformly from the rest K-1 entities
  **end if**
  calculate the acceptance ratio $\alpha$
  apply the proposal with probability $\alpha$
**end procedure**

**Algorithm 3** Smart Merge and Dumb Split proposal

**procedure** SMART MERGE DUMB SPLIT
  choose a move type: $type \sim (split, merge)$
  **if** type==merge **then** ▷ smart merge
    choose one entity $E_i$ uniformly from K entities
    choose another entity

$$E_j \sim f_{merge}(E_j|E_i)$$

$$f_{merge}(E_j|E_i) \propto P(\mathbf{m}_{E_i,E_j}, E_j|E_i)$$

  **else** ▷ dumb split
    choose one entity $E_k$ uniformly from K entities
    create two new empty entities $E'_1$ and $E'_2$
    assign each mention in $E_k$ sequentially to $E'_1$ or $E'_2$ with equal probability
  **end if**
  calculate the acceptance ratio $\alpha$
  apply the proposal with probability $\alpha$
**end procedure**

Table 1.

Table 1: Splitting one entity into two entities sequentially

| steps | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $E'_1$ | | A | AA | AA | AA | AA | AA |
| $E'_2$ | | | | B | BB | BBC | BBCC |
| P | | 0.5 | 0.6 | 0.625 | 0.714 | 0.5 | 0.714 |

Two new created entities are empty in the beginning. During each step, the allocation probability for each mention $m_i$ is calculated by:

$$\begin{aligned} P(E'_1|m_i) &\propto \frac{\alpha_{1,l'} + n_{1,l'}}{\sum_{l' \in L}(\alpha_{1,l'} + n_{1,l'})} \\ P(E'_2|m_i) &\propto \frac{\alpha_{2,l} + n_{2,l}}{\sum_{l' \in L}(\alpha_{2,l'} + n_{2,l'})} \end{aligned} \quad (5)$$

where $\alpha_{1,l}$ and $\alpha_{2,l}$ are the Dirichlet priors for the token $l$ (the mention $m_i = l$, which is A, B or C in this case) being assigned to entity $E'_1$ and $E'_2$, $n_{1,l}$ and $n_{2,l}$ are current observed counts of token $l$ assigned to $E'_1$ and $E'_2$ (counts are updated during each step). The denominator sums up the prior and observed counts for all possible tokens (A, B and C in this case). The probability in each step is given in the table taking all $\alpha_{1,l}$=$\alpha_{2,l}$=1 as example (smaller alpha makes this procedure more discriminating). The probability of this allocation procedure is then a product of the probability in each step. This procedure avoids the time-consuming Restricted Gibbs sampling adopted by Jain and Neal [2004]. [Dahl, 2003] proposed a similar sequential procedure but started by creating two new entities with two random mentions, which causes an initial bias when these two random mentions are supposed to be associated with the same entity.

The reverse dumb merge proposal would rather generate random merges. It picks one entity uniformly from K entities and then picks another from the remaining K-1 entities. The probability of this reverse proposal is then $2/K(K-1)$ (two orders of choosing these two entities).

If a dumb merge proposal is chosen in the beginning, the choice of two entities will have the probability $1/K(K-1)$. Then we need to know the probability of the split proposal that reverses this move, i.e., allocates the mentions exactly into the two given sets. The order of mentions during allocation influences the final probability and the different probabilities from all possible orders are supposed to be summed up, which is not really feasible in practice. Dahl [2003] applied a random permutation on the order of mentions, which may be critical to correctness of MCMC methods since in this way we are obtaining a random probability for the reverse split when merge is proposed and it may not correspond to the exact probability of the inverse move. In our case, we fix a unique order for all mentions so that there is only one way of applying the procedure thus only one possible probability value for the same split results, either the real split procedure or the imaged reverse one. This unique order is chosen in an arbitrary way. The choice of any particular order has no influence on the inference but the same order should be kept all along the experiments.

**Smart merge/dumb split proposal** Algorithm 3 begins from making the random choice between merge and split as well. If smart merge proposal is picked, it will first choose

one entity $E_i$ randomly from K entities. The choice of the second entity $E_j$ is based on how likely it is when merged with $E_i$. The entity $E_j$ is draw from a distribution given by function $f_{merge}(E_j|E_i)$, which is proportional to the likelihood of this resulting merge of $E_j$ to $E_i$. The likelihood of the merge is calculated by:

$$P(\mathbf{m}_{E_i,E_j}, E_j|E_i) = \frac{B(\boldsymbol{\alpha}_i + \mathbf{n}_i + \mathbf{n}_j)}{B(\boldsymbol{\alpha}_i)} \quad (6)$$

where $\mathbf{m}_{E_i,E_j}$ refers to all mentions assigned to $E_i$ and $E_j$, $\boldsymbol{\alpha}_i$ is vector for Dirichlet prior, and $\mathbf{n}_i + \mathbf{n}_j$ refers to the vector for the observed counts of each term in $\mathbf{m}_{E_i,E_j}$.

The reverse dumb split proposal chooses one entity $E_k$ uniformly from K entities and allocates each mention randomly into two new created entities, the probability of which is $1/(K \cdot 2^n)$, where n is the number of mention assigned to entity $E_k$.

If the dumb split proposal is chosen, it generates a random split with the same probability $1/(K \cdot 2^n)$. As for the reverse smart merge proposal, we need to consider the merge in two different orders, namely $f_{merge}(E_j|E_i)$ and $f_{merge}(E_i|E_j)$, where $f_{merge}(E_i|E_j)$ is proportional to $P(\mathbf{m}_{E_j,E_i}, E_i|E_j)$ which can be calculated similarly to Formula 6.

# 5 EXPERIMENTS WITH SDDS

## 5.1 Applying SDDS to EMM

We applied the SDDS algorithm to the entity/mention model. During each inference step, the SDDS sampler chooses uniformly from either smart-split/dumb-merge proposal or dumb-split/smart-merge proposal; this is then complemented by one single Gibbs sampling step.[2] It is worth emphasizing again that detailed balance is satisfied when each sub-kernel fulfills the detailed balance condition individually. Three other algorithms are tested for comparison. The first is a random mixing (RM) sampler inspired from [Pasula *et al.*, 2003]. It works by choosing two random entities (which could be the same one) and then distributing corresponding mentions into two new created entities by a randomly chosen split point. The second is the RGSM sampler [Jain and Neal, 2004].[3] The third is a smart-smart (SS) sampler which has the same smart split and merge moves as SDDS but lacks the paired dumb moves.

The simulated data sets use 10 different tokens (A, B, C, etc.). Different configurations were tested, including different data set sizes ($N$=100, 200, 500) and different initial entity numbers ($K_0$=1, 5, 10, 20). All Dirichlet priors $\alpha$ are set to 0.001. For the Log-normal prior on the number of entities $K$, the location parameter is set to 0.5 (10 entities) and scale parameter is set to 1. Experiments were run for 10K/50K/200K iterations, with a time-out at 10 hours.

We are interested in the posterior distribution of $K$, the number of entities, given the available evidence. We illustrate the evolution of the expected value of $K$ for the various algorithms as a function of the number of iterations, for different values of $N$ and $K_0$. We also give the acceptance rates and time needed for each iteration for comparison.

**Posterior distribution of entity numbers** The posterior distributions of entity numbers are analyzed by the mean of the value K during iterations. First of all, we fix the initial value $K_0$=5 to compare results of these four different algorithms for different data sizes, as shown in Figure 3. We can see that for $N = 100$ and $N$=200, all samplers except random mixing sampler can converge to the correct posterior K=10. However, when a larger data set $N$=500 is used, the RGSM sampler fails to have successful split or merge therefore the value K is trapped at initial value. The SS sampler is capable of applying several effective splits whereas the merge proposal does not work well because of the reason we discussed in Section 4.1.

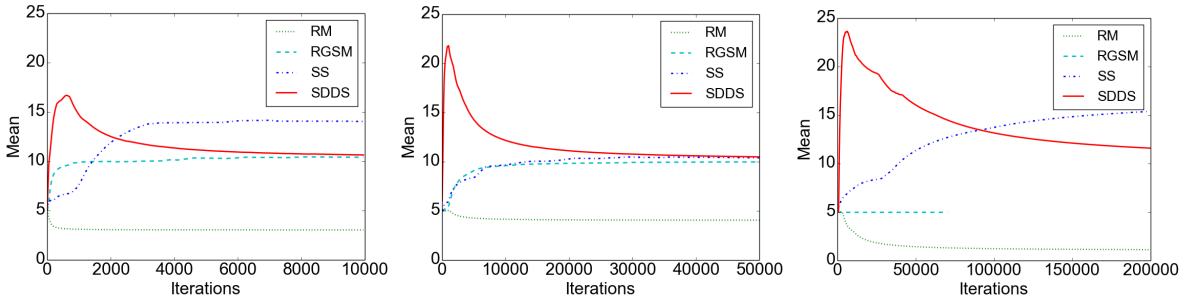
For the data set $N$=500, we have also analyzed the sensitivity of the sampler to initial value $K_0$, as illustrated in Figure 4. It is easy to observe that RGSM sampler is always trapped in initial $K_0$ in this case. The SS sampler can generate well-assigned mentions for entities during smart split procedure, whereas it can not converge to the true posterior. It is interesting to see from the results of Random Mixing sampler that it works well when the initial value is twice as large as the true value. In fact, in the RM sampler, the random split procedure encourages the acceptance of merge proposals. However, the random split proposals themselves are not likely to be accepted because they generate very low-probability states. The SDDS sampler outperforms all the others, converging to the true posterior regardless of the initial value.

It is worth mentioning that the inferences start from random allocations, which would generate an ensemble of messed up entities. It is not really useful to merge messed up entities. The acceptance rates for merging messed up entities should be very low as well. It is therefore logical to observe many accepted splits in the beginning of the inference of SDDS algorithm. SDDS algorithm generates more well-formed entities and then yields higher probabilities for merging them.

**Acceptance rates** The relation between the size of data set and the acceptance rates for both split and merge are shown in Figure 5. We can observe that, for the data set

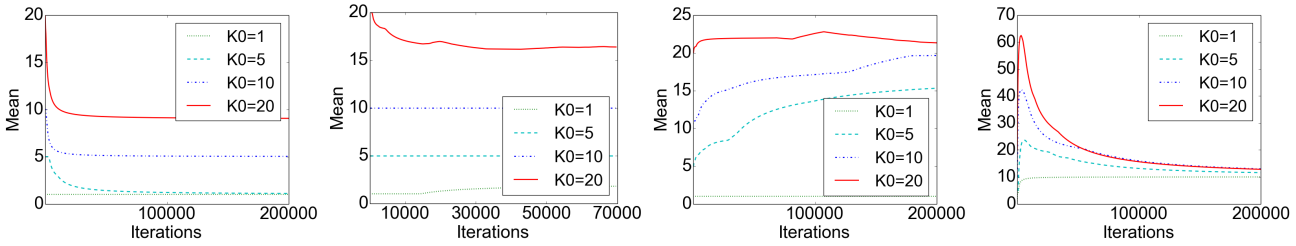
[2]The different ways of combining a designed sampler with Gibbs sampler is an issue to be investigated; we adopted this configuration for all tested algorithms for comparison.

[3]The stable version (5, 1, 1) is adopted for comparison, which contains 5 intermediate Restricted Gibbs steps inside one MH step and then one complete Gibbs sampling.




(a) $N$=100 (b) $N$=200 (c) $N$=500

Figure 3: The mean of K during iteration for different data size, with the initial $K_0$=5




(a) Random Mixing (b) Restricted Gibbs Split-Merge (c) Smart Split Smart Merge (d) Smart-Dumb Dumb-Smart

Figure 4: The mean of K during iteration for different initial $K_0$=1,5,10,20, with the data size N=500

$N$=100, RGSM sampler has high acceptance rates for both split (7.8%) and merge (2.1%). However, when the data size is $N$=200, the acceptance rates are decreased largely, only 1.1% for split and 0.08% for merge. When the data size grows to $N$=500, almost no effective split or merge is happening. On the contrary, the smart proposals in SDDS algorithm, either smart split or smart merge, maintain satisfying acceptance rates even for the data set $N$=500, 1.9% for smart split and 4.8% for smart merge. For the SS sampler, the drop of acceptance rates is similar to RGSM sampler since there is no dumb proposals to support their corresponding smart proposals.

**Time per iteration** As we stated previously, the allocation procedure in our smart split proposal is less time-consuming than the Restricted Gibbs sampling based split proposal. The performance of the running time per iteration is shown for each algorithm in Figure 5. For the RGSM sampler, the time spent per iteration grows quickly with the data size, whereas the time for SDDS algorithm remains stable. In the case of $N$=500, RGSM sampler takes 488.63 milliseconds per step while SDDS algorithm takes only 10.02 milliseconds per step. The time per iteration for Random Mixture sampler and that for SS sampler (which are not show in the figure) is in the same scale of SDDS sampler and stays stable for different data sizes.

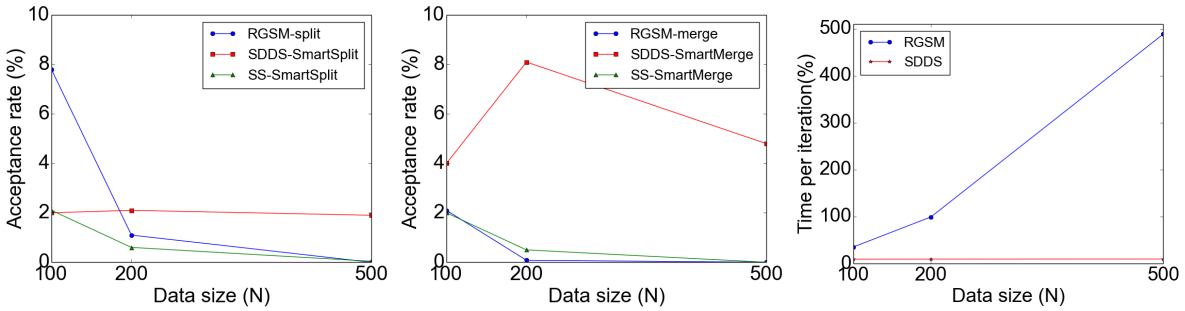
### 5.2 Applying SDDS to conjugate Dirichlet Process Mixture Model

RGSM algorithm is originally proposed for Dirichlet Process Mixture Model (DPMM). When conjugate priors are used, the Gibbs sampling procedure can be easily constructed for DPMM. A particular Gibbs sampling method is adapted in this context of DPMM model so that the Gibbs sampler can create new components [Neal, 1992]. The proposed SDDS algorithm is also applied to DPMM and is then compared to the Gibbs sampler and RGSM sampler.

The experiments have been done with high dimensional Bernoulli data. Given the independently and identically distributed data set $\mathbf{y} = (y_1, y_2, ..., y_N)$ , each observation $y_i$ has $m$ Bernoulli attributes, $(y_{i1}, y_{i2}, ..., y_{im})$. Given the component $c_i$ each item $y_i$ belongs to, its attributes are independent of each other. The mixture components are considered as the latent class that produces the observed data.

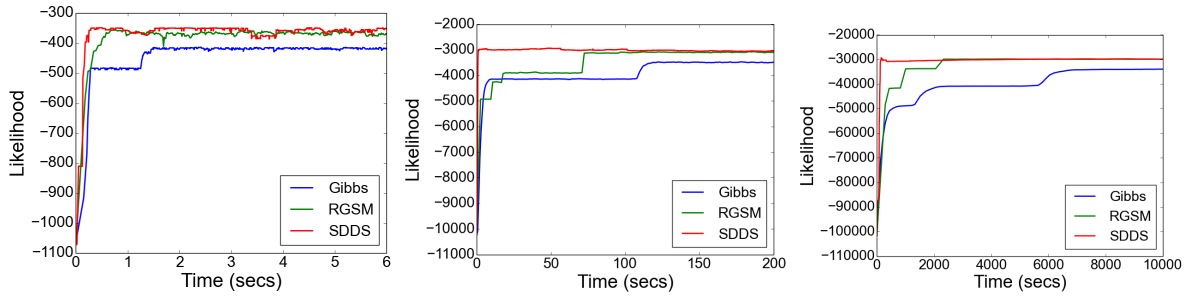
The simulated data for our experiments are generated in the same way as Jain and Neal [2004] did for one high-dimensional data set: 5 components with attribute dimension 15. Experiments are run for different sizes, $N$=100, 1K, and 10K. The Dirichlet process prior and the Beta prior for attributes are respectively set to 1 and 0.1. (See Jain and Neal [2004] for further details on this model).

;Jain and Neal [2004] have demonstrated their results by plotting the traces of the five ;highest-wweight components




(a) Acceptance rates for split (b) Acceptance rates for merge (c) Time per Iteration

Figure 5: The differences of acceptance rates and time per iteration for different data sizes




(a) Data size N=100 (b) Data size N=1K (c) Data size N=10K

Figure 6: Comparison of the evolution of likelihood during the time for different data sizes

to verify if their algorithms can cover most data and detect ;five components. We have provided the same plots for the SDDS sampler. We observed the same relative performance for SDDS and RGSM on the DPMM as for the EMM, in terms of time per iteration and acceptance rates (results not shown here). We also compared the evolution of the likelihoods for SDDS, RGSM, and Gibbs as a function of running time (Figure 6). We see that, for $N$=100, RGSM arrives at high-probability states about 0.5 second after the SDDS algorithm does. When $N$=1K, SDDS gains 70 seconds over RGSM. When $N$=10K, SDDS outperforms RGSM by more than 2000 seconds. We conclude from this limited sample of runs that the advantage of SDDS over RGSM increases with data set size.

## 6 CONCLUSION

We have described the SDDS algorithm, which achieves efficient split–merge inference by combining smart and dumb proposals. The idea is illustrated for the entity/mention model. For the smart split proposal, we proposed a fast and exact way of generating splits that are consistent with data. Experiments on the entity/mention model and Dirichlet process mixture models suggest that SDDS algorithm mixes faster than previously known algorithms, and that the advantage increases with data set size.

## References

Gilles Celeux, Merrilee Hurn, and Christian P. Robert. Computational and inferential difficulties with mixture posterior distributions. *Journal of the American Statistical Association*, 2000.

J Chang and J. W. Fisher. Parallel sampling of dp mixture models using sub-clusters splits. In *NIPS*, 2013.

David B Dahl. An improved merge-split sampler for conjugate dirichlet process mixture models. Technical report, University of Wisconsin - Madison, 2003.

Andrew Gelman. Iterative and non-iterative simulation algorithms. In *Computing Science and Statistics: Proceedings of the 13th Symposium on the Interface*, 1992.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1984.

W K Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 1970.

Michael Hughes, Emily Fox, and Erik Sudderth. Effective split-merge Monte Carlo methods for nonparametric models of sequential data. In *NIPS*, 2012.

Sonia Jain and Radford Neal. A split-merge Markov chain Monte Carlo procedure for the dirichlet process mixture

model. *Journal of Computational and Graphical Statistics*, 2004.

Zia Khan, T. Balch, and F. Dellaert. Multitarget tracking with split and merged measurements. In *CVPR*, 2005.

N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 1953.

Radford M. Neal. Bayesian mixture modeling. In *Proceedings of the 11th International Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, 1992.

Mathias Niepert and Pedro Domingos. Exchangeable variable models. In *ICML*, 2014.

Hanna Pasula, Stuart Russell, Michael Ostland, and Ya'acov Ritov. Tracking many objects with many sensors. In *IJCAI*, 1999.

Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *NIPS*. MIT Press, 2003.

Santu Rana, Dinh Phung, and Svetha Venkatesh. Split-merge augmented gibbs sampling for hierarchical dirichlet processes. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2013.

Luke Tierney. Markov chains for exploring posterior distributions. *Annals of Statistics*, 1994.

Chong Wang and David M. Blei. A split-merge MCMC algorithm for the hierarchical dirichlet process. *CoRR*, 2012.

Sinead Williamson, Avinava Dubey, and Eric P. Xing. Parallel Markov chain Monte Carlo for nonparametric mixture models. In *ICML*, 2013.