Fast Parallel Randomized Algorithm for Nonnegative Matrix Factorization with KL Divergence for Large Sparse Datasets

Duy-Khuong Nguyen and Tu-Bao Ho

Abstract—Nonnegative Matrix Factorization (NMF) with Kullback-Leibler Divergence (NMF-KL) is one of the most significant NMF problems and equivalent to Probabilistic Latent Semantic Indexing (PLSI), which has been successfully applied in many applications. For sparse count data, a Poisson distribution and KL divergence provide sparse models and sparse representation, which describe the random variation better than a normal distribution and Frobenius norm. Specially, sparse models provide more concise understanding of the appearance of attributes over latent components, while sparse representation provides concise interpretability of the contribution of latent components over instances. However, minimizing NMF with KL divergence is much more difficult than minimizing NMF with Frobenius norm; and sparse models, sparse representation and fast algorithms for large sparse datasets are still challenges for NMF with KL divergence. In this paper, we propose a fast parallel randomized coordinate descent algorithm having fast convergence for large sparse datasets to archive sparse models and sparse representation. The proposed algorithm’s experimental results overperform the current studies’ ones in this problem.

Index Terms—Nonnegative matrix factorization, kullback-leibler divergence, sparse models, and sparse representation.

I. INTRODUCTION

The development of technology has been generating big datasets of count sparse data such as documents and social network data, which requires fast effective algorithms to manage this huge amount of information. One of these tools is nonnegative matrix factorization (NMF) with KL divergence, which is proved to be equivalent with Latent Semantic Indexing (PLSI) [1].

NMF is a powerful linear technique to reduce dimension and to extract latent topics, which can be readily interpreted to explain phenomenon in science [2]-[4]. NMF makes post-processing algorithms such as classification and information retrieval faster and more effective. In addition, latent factors extracted by NMF can be more concisely interpreted than other linear methods such as PCA and ICA [4]. In addition, NMF is flexible with numerous divergences to adapt a large number of real applications [5], [6].

For sparse count data, NMF with KL divergence and a Poisson distribution may provide sparse models and sparse representation describing better the random variation rather than NMF with Frobenius norm and a normal distribution [7]. For example, the appearance of words over latent topics and topics over documents should be sparse. However, achieving sparse models and sparse representation is still a major challenge because minimizing NMF with KL divergence is much more difficult than NMF with Frobenius norm [8].

In the NMF-KL problem, a given nonnegative data matrix \( V \in \mathbb{R}_+^{nm} \) must be factorized into a product of two nonnegative matrices, namely a latent component matrix \( W \in \mathbb{R}_+^{nm} \) and a representation matrix \( F \in \mathbb{R}_+^{nm} \), where \( n \) is the dimension of a data instance, \( m \) is the number of data instances, and \( r \) is the number of latent components or latent factors. The quality of this factorization is controlled by the objective function with KL divergence as follows:

\[
D(V \| W^T F) = \sum_{i=1}^{m} \sum_{j=1}^{n} (V_{ij} \log \frac{W_{ij}}{(W^T F)_j} - V_{ij} - (W^T F)_j) \tag{1}
\]

In the general form of \( L_1 \) and \( L_2 \) regularization variants, the objective function is written as follows:

\[
D(V \| W^T F) + \frac{\alpha_1}{2} \| W \|_F^2 + \frac{\alpha_2}{2} \| F \|_F^2 + \beta \| W \|_1 + \beta \| F \|_1 \tag{2}
\]

NMF with KL divergence has been widely applied in many applications for dense datasets. For example, spatially localized, parts-based subspace representation of visual patterns is learned by local non-negative matrix factorization with a localization constraint (LNMF) [9]. In another study, multiple hidden sound objects from a single channel auditory scene in the magnitude spectrum domain can be extracted by NMF with KL divergence [10]. In addition, two speakers in a single channel recording can be separated by NMF with KL divergence and \( L_1 \) regularization on \( F \) [11].

However, the existing algorithms for NMF with KL divergence (NMF-KL) are extremely time-consuming for large count sparse datasets. Originally, Lee and Seung, 2001 [12] proposed the first multiple update iterative algorithm based on gradient methods for NMF-KL. Nevertheless, this technique is simple and ineffective because it requires a large number of iterations, and it ignores negative effects of
nonnegative constraints. In addition, gradient methods have slow convergence for complicated logarithmic functions like KL divergence. Subsequently, Cho-Jui & Inderjit, 2011 [13] proposed a cycle coordinate descent algorithm having low complexity of one variable update. However, this method contains several limitations: first, it computes and stores the dense product matrix \( W^T F \) although \( V \) is sparse; second, the update of \( W^T F \) for the change of each cell in \( W \) and \( F \) is considerably complicated, which leads to practical difficulties in parallel and distributed computation; finally, the sparsity of data is not considered, while large datasets are often highly sparse.

In comparison with NMF with Frobenius norm, NMF with KL divergence is much more complicated because updating one variable will influence derivatives of other variables; this computation is extremely expensive. Hence, it is difficult to employ fast algorithms having multiple variable updates, which limits the number of effective methods.

In this paper, we propose a new advanced version of coordinate descent methods with significant modifications for large sparse datasets. Regarding the contributions of this paper, we:

- Propose a fast sparse randomized coordinate descent algorithm using limited internal memory for nonnegative matrix factorization for huge sparse datasets, the full matrix of which can not be stored in the internal memory. In this optimization algorithm, variables are randomly selected with uniform sampling to balance the order priority of variables. Moreover, the proposed algorithm effectively utilizes the sparsity of data, models and representation matrices to improve its performance. Hence, the proposed algorithm can be considered an advanced version of cycle coordinate descent for large sparse datasets proposed in [13].
- Design parallel algorithms for combinational variants of \( L_1 \) and \( L_2 \) regularizations.
- Indicate that the proposed algorithm using limited memory can fast attain sparse models, sparse representation, and fast convergence by experimental evaluations, which is a significant milestone in this research problem for large sparse datasets.

The rest of the paper is organized as follows. Section II presents the proposed algorithms. The theoretical analysis of convergence and complexity is discussed in Section III. Section IV shows the experimental results, and Section V summarizes the main contributions of this paper and discussion.

### II. PROPOSED ALGORITHM

In this section, we propose a fast sparse randomized coordinate descent parallel algorithm for nonnegative sparse matrix factorization on Kullback-Leibler divergence. We employ a multiple iterative update algorithm like EM algorithm, see Algorithm 1, because the objective function \( D(V \parallel W^T F) \) is a non-convex function although it is a convex function when fixing one of two matrices \( W^T \) and \( F \). This algorithm contains a while loop containing two main steps: the first one is to optimize the objective function by \( F \) when fixing \( W^T \); and the other one is to optimize the objective function by \( W \) when fixing \( F \). Furthermore, in this algorithm, we need to minimize Function 3, the decomposed elements of which can be independently optimized in Algorithm 1:

\[
D(V \parallel W^T F) = \sum_{i=1}^m D(V_i \parallel W_i^T F_i) = \sum_{i=1}^m D(V_i^T \parallel F_i^T W_i)
\]  (3)

Specially, a number of optimization problems \( D(V_i \parallel W_i^T F_i) \) or \( D(V_i^T \parallel F_i^T W_i) \) in Algorithm 1 with the form \( D(v \parallel Ax) \) can be independently and simultaneously solved by Algorithm 2. In this paper, we concern combinational variants of NMF KL divergence with \( L_1 \) and \( L_2 \) regularizations in the general formula, Function 4:

\[
f(x) = D(x \parallel Ax) = \sum_{i=1}^k (\alpha v_i - \nu_i + \Delta(x)) + \frac{\alpha}{2}||x||_2^2 + \beta \|x\|_1
\]  (4)

where \( v \in R^n \), \( A \in R^{n \times r} \), and \( x \in R^r \).

### Algorithm 1: Iterative multiplicative update

**Input:** \( V \in R^{n \times k} \), \( r \), and \( \alpha, \beta, \gamma, \beta_2 \geq 0 \)

**Output:** \( W \in R^{n \times r} \), \( F \in R^{r \times m} \).

1. **begin**
   2. Randomize \( W \in R^{n \times r} \);
   3. Randomize \( F \in R^{r \times m} \);
   4. while convergence condition is not satisfied do
      5. \( ids = \) a randomized ordered set of values \( \{1, 2, \ldots, r\} \);
      6. \( sumW = W1' \);
      7. \( i^{\text{th}}\) Optimizing the objective function by \( F \) when fixing \( W^{\text{old}} \);
   8. for \( j = 1 \) to \( m \) do
      9. \( i^{\text{th}}\) Call Algorithm 2 in parallel \( F \);
      10. \( F_j^{k+1} = \) Algorithm 2 \((V_j, W^T, sumW, F_j, \alpha_2, \beta_2, ids)\);
      11. \( sumF = F1' \);
      12. \( i^{\text{th}}\) Optimizing the objective function by \( W \) when fixing \( F^{\text{old}} \);
   13. for \( i = 1 \) to \( n \) do
      14. \( i^{\text{th}}\) Call Algorithm 2 in parallel \( V \);
      15. \( W_i^{k+1} = \) Algorithm 2 \((V_i^T, F^T, sumW, W_i, \alpha_2, \beta_2, ids)\);
   16. return \((W^{k+1})^T, F^{k+1}\);

Because the vector \( v \) is given, minimizing Function 4 is equivalent to minimizing Function 5:

\[
f(x) = D(x \parallel Ax) = \sum_{i=1}^k (\alpha v_i - \nu_i + \Delta(x)) + \frac{\alpha}{2}||x||_2^2 + \beta \|x\|_1
\]  (5)

From Equation 5, the first and second derivative of the variable \( x_k \) are computed by Formula 6:

\[
\nabla f_k = \frac{\partial f_k}{\partial x_k} = -\sum_{i=1}^n v_i \frac{A_k}{A_k} + \sum_{i=1}^n A_k + \alpha x_k + \beta
\]

\[
\nabla^2 f_k = \frac{\partial^2 f_k}{\partial x_k^2} = \sum_{i=1}^n (A_k)^2 + \alpha
\]  (6)

Based on Formula 6, we have several significant remarks:

- One update of \( x_k \) changes all elements of \( Ax \), which are
under the denominators of fractions. Hence, it is difficult to employ fast algorithms having simultaneous updates of multiple variables because it will require heavy computation. Hence, we employ coordinate descent methods to reduce the complexity of each update, and to avoid negative effects of nonnegative constraints.

- One update of $x_k$ has complexity of maintaining $\nabla f_k$ and $\nabla^2 f_{sk}$ as $O(k + \text{nnz}(v) + \text{nnz}(v)) \approx k + \text{nnz}(v)$ when $\sum_{i=1}^n A_{sk}$ is computed in advance. Exactly, this update employs $O(k + \text{nnz}(v))$ of multiple and addition operators; and $O(\text{nnz}(v))$ of divide operators; where $\text{nnz}(v)$ is the number of is the number of non-zero elements in the vector $v$. Hence, for sparse datasets, the number of operators can be negligible.

- The used internal memory of Algorithm 1 and Algorithm 2 is $O(\text{nnz}(V) + \text{size}(W) + \text{size}(F)) \approx \text{nnz}(V) + (n + m) r$ where $\text{nnz}(V)$ is the number of non-zero elements in the given matrix $V$, which is much smaller than $O(mn + (n + m) r)$ for the existing algorithms [12], [13].

Hence, Algorithm 2 employs a coordinate descent algorithm based on projected Newton methods with quadratic approximation in Algorithm 2 is to optimize Function 5. Specially, because Function 5 is convex, a coordinate descent algorithm based on projected Newton method [14] with quadratic approximation is employed to iteratively update with the nonnegative a lower bound as follows:

$$x_k = \max(0, x_k - \frac{\nabla f_k}{\nabla^2 f_{sk}})$$

Considering the limited internal memory and the sparsity of $x$, we maintain $W^TF$ via computing $Ax$ instead of storing the dense matrix $W^TF$ for the following reasons:

- The internal memory requirement will significantly decrease, so the proposed algorithm can stably run on limited internal memory machines.

- The complexity of computing $Ax$ is always smaller than the complexity of computing and maintaining $\nabla f_k$ and $\nabla^2 f_{sk}$, so it does not cause the computation more complicated.

- The updating $Ax = Ax + \Delta x A_{sk}$ as the adding with a scale of two vectors utilizes the speed of CPU cache because of accessing consecutive memory cells.

- The recomputing helps remove the complexity of maintaining the dense product matrix $W^TF$ as in [12], [13], which is certainly considerable because this maintenance accesses memory cells far together and does not utilize CPU cache.

In summary, in comparison with the original coordinate algorithm [13] for NMK-KL, the proposed algorithm involve significant improvements as follows:

- Randomize the order of variables to optimize the objective function in Algorithm 2. Hence, the proposed algorithm can balance the order priority of variables.

- Remove duplicated computation of maintaining derivatives $\nabla f_k$ and $\nabla^2 f_{sk}$ by computing common elements $\text{sum} = W^n$ and $\text{sum}F = F^n$ in advance, which led to that the complexity of computing $\nabla f_k$ and $\nabla^2 f_{sk}$ only depends on the sparsity of data.

- Effectively utilize the sparsity of $W$ and $F$ to reduce the running time of computing $Ax$.

- Recompute $Ax$ but remove the maintenance of th dense matrix product $W^TF$. Hence, the proposed algorithm stably run on the limited internal memory systems with the required memory size $O(\text{nnz}(V) + \text{size}(W) + \text{size}(F)) = \text{nnz}(V) + (m + n) r$, which is much smaller than $O(mn + (n + m) r)$ for the existing algorithms [12], [13].

### III. THEORETICAL ANALYSIS

In this section, we analyze the convergence and complexity of Algorithm 1 and Algorithm 2.

In comparison with the previous algorithm of Hsieh & Dhillon, 2011 [13], the proposed algorithm has significant modifications for large sparse datasets by means of adding the order randomization of indexes and utilizing the sparsity of data $V$, model $W$, and instance representation $F$. These modifications do not affect on the convergence guarantee of algorithm. Hence, based on Theorem 1 in Hsieh & Dhillon, 2011 [13], Algorithm 2 converges to the global minimum of $f(x)$. Furthermore, based on Theorem 3 in Hsieh & Dhillon, 2011 [13], Algorithm 1 using Algorithm 2 will converge to a stationary point. In practice, we set $\epsilon = 0.1$ in Algorithm 2, which is more precise than $\epsilon = 0.5$ in Hsieh & Dhillon, 2011 [13].

Concerning the complexity of Algorithm 2, based on the remarks in Section II, we have Theorem 1. Furthermore, because KL divergence is a convex function over one variable and the nonnegative domain, and project Newton methods with quadratic approximation for convex functions have superlinear rate of convergence [14], [15], the average number of iterations $l$ is small.

**Theorem 1.** The complexity of Algorithm 2 is $O(n, \text{nnz}(r) + \tilde{r}(r + n + \text{nnz}(n)))$, where $\text{nnz}(r)$ is the number of non-zero elements in $x$, $\text{nnz}(n)$ is the number of non-zero elements in $v$, and $\tilde{r}$ is the average number of iterations. Then, the complexity of a while iteration in Algorithm 1 is $O(\tilde{r}(mn + (m + n) r^2))$.

Proof. Consider the major computation in Algorithm 2, based on Formula 6, we have:

- The complexity of computing $Ax$ in Line 2 is $O(n, \text{nnz}(r))$.

- The complexity of computing $\nabla f_k$ and $\nabla^2 f_{sk}$ in Line 4 is $O(r + \text{nnz}(n))$ because $Ax$ and $\text{sumA}$ are computed in advance.
The complexity of updating $\nabla f_i$ and $\nabla^2 f_{i,a}$ in Line 12 is $O(r + n + nnz(n))$ because only one dimension of vector $x$ is changed. Hence, the complexity of Algorithm 2 is $O(n.nnz(r) + \tilde{t}r(r + n + nnz(n)))$.

In addition, the complexity of computing $\text{sum}W$ and $\text{sum}F$ is $O(n + m)r$. Hence, the complexity of a while iteration in Algorithm 1 is $O((m+n)r + mn.nnz(r) + \tilde{t}mr(r + n + nnz(n)) \approx (m+n)r + \tilde{t}(mn r + (m+n)r^2)) \approx \tilde{t}(mn r + (m+n)r^2)$. Therefore, we have Theorem 1.

For large sparse datasets, $m, n \gg r \Rightarrow O(\tilde{t}(mn r + (m+n)r^2)) \approx \tilde{t}(mn r)$. This complexity is raised by the operators $Ax = Ax + \Delta x A_t$ in Algorithm 2. To reduce the running time of these operators, $A_k$ must be stored in an array to utilize CPU cache memory by accessing continuous memory cells of $Ax$ and $A_k$.

IV. EXPERIMENTAL EVALUATION

In this section, we investigate the effectiveness of the proposed algorithm via convergence and sparsity. Specially, we compare the proposed algorithm Sparse Randomized Coordinate Descent (SRCD) with state-of-the-art algorithms as follows:

- Multiplicative Update (MU) [12]: This algorithm is the original method for NMF with KL divergence.
- Cycle Coordinate Descent (CCD) [13]: This algorithm has the current fastest convergence because it has very low complexity of each update for one variable.

Datasets: To investigate the effectiveness of the algorithms compared, the 4 sparse datasets used are shown in Table I. The dataset Digit is downloaded from http://yann.lecun.com/, and the other tf-idf datasets Reuters21578, TDT2, and RCV1 4Class are downloaded from http://www.cad.zju.edu.cn/.

<table>
<thead>
<tr>
<th>Dataset (V)</th>
<th>n</th>
<th>m</th>
<th>nnz(V)</th>
<th>Sparsity(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>784</td>
<td>60,000</td>
<td>8,994,156</td>
<td>80.880</td>
</tr>
<tr>
<td>Reuters21578</td>
<td>8,293</td>
<td>18,933</td>
<td>389,455</td>
<td>99.752</td>
</tr>
<tr>
<td>TDT2</td>
<td>9,394</td>
<td>36,771</td>
<td>1,224,135</td>
<td>99.645</td>
</tr>
<tr>
<td>RCV1 4Class</td>
<td>9,625</td>
<td>29,992</td>
<td>730,879</td>
<td>99.746</td>
</tr>
</tbody>
</table>

Environment settings: We develop the proposed algorithm SRCD in Matlab with embedded code C++ to compare them with other algorithms. We set system parameters to use only 1 CPU for Matlab and the IO time is excluded in the machine Mac Pro 8-Core Intel Xeon E5 3.0 GHz 32GB. In addition, the initial matrices $W_0$ and $F_0$ are set to the same values. The source code will be published on our homepage http://khuongnd.appspot.com/.

A. Convergence

In this section, we investigate the convergence of the objective value $D(V \| W^T F)$ versus running time by running all the compared algorithms on the four datasets with two numbers of latent factors $r = 10$ and $r = 20$. The experimental results are depicted in Fig. 1 and Fig. 2. From these figures, we realize two significant observations as follows:

- The proposed algorithm (SRCD) has much faster convergence than the algorithms CCD and MU.
- The sparser the datasets are, the greater the distinction between the convergence of the algorithm SRCD and the other algorithms CCD and MU is. Specially, for Digits with 81% sparsity, the algorithm SRCD’s convergence is lightly faster than the convergence of the algorithms CCD and MU. However, for three more sparse datasets Reuters21578, TDT2, and RCV1 4Class with above 99% sparsity, the distance between these convergence speeds is readily apparent.

B. Sparsity of Factor Matrices

Concerning the sparsity of factor matrices $W$ and $F$, the
algorithms CCD and MU does not utilize the sparsity of factor matrices. Hence, these algorithms add a small number into these factor matrices to obtain convenience in processing special numerical cases. Hence, the sparsity of factor matrices \( W \) and \( F \) for the algorithms CCD and MU both are 0%.

Although this processing may not affect other post-processing tasks such as classification and information retrieval, it will reduce the performance of these algorithms. The sparsity of \((W, F)\) of the proposed algorithm’s results is shown in Table II. These results clearly indicate that the sparse model \( W \) and the sparse representation \( F \) are attained.

The results also explain why the proposed algorithm runs very fast on the sparse datasets Reuters21578, TDT2 and RCV1 4Class, when it can obtain highly sparse models and sparse representation in these highly sparse datasets.

### TABLE II: SPARSKITY (% OF \((W, F)\) FOR THE ALGORITHM SRCD’S RESULTS

<table>
<thead>
<tr>
<th>Digits</th>
<th>Reuters21578</th>
<th>TDT2</th>
<th>RCV1 4Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r=10)</td>
<td>(74.3, 49.2)</td>
<td>(75.6, 71.6)</td>
<td>(68.5, 71.3)</td>
</tr>
<tr>
<td>(r=20)</td>
<td>(87.8, 49.7)</td>
<td>(84.2, 80.4)</td>
<td>(78.6, 81.1)</td>
</tr>
</tbody>
</table>

### C. Used Internal Memory

Table III shows the internal memory used by algorithms. From the table, we have two significant observations:
- For the dense dataset Digits, the proposed algorithm SRCD uses more internal memory than the other algorithms CCD because a considerable amount of memory is used for the indexing of matrices.
- For the sparse datasets Reuters21578, TDT2, and RCV1 4Class, the internal memory for SRCD is remarkably smaller than the internal one for MU and CCD.

These results indicate that we can conduct the proposed algorithm for huge sparse datasets with a limited internal memory is stable.

### TABLE II: USED INTERNAL MEMORY (GB) FOR \(r = 10\)

<table>
<thead>
<tr>
<th>Datasets</th>
<th>MU</th>
<th>CCD</th>
<th>SRCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>1.89</td>
<td>0.85</td>
<td>1.76</td>
</tr>
<tr>
<td>Reuters21578</td>
<td>5.88</td>
<td>2.46</td>
<td>0.17</td>
</tr>
<tr>
<td>TDT2</td>
<td>11.51</td>
<td>5.29</td>
<td>0.30</td>
</tr>
<tr>
<td>RCV1 4Class</td>
<td>9.73</td>
<td>4.43</td>
<td>0.23</td>
</tr>
</tbody>
</table>

### D. Running on Large Datasets

This section investigates running the proposed algorithm on large datasets with different settings. Fig. 3 shows the running time of Algorithm SRCD for 100 iterations with different number of latent component using 1 thread. Clearly, the running time linearly increases, which fits the theoretical analyses about fast convergence and linear complexity for large sparse datasets in Section III. Furthermore, concerning the parallel algorithm, the running time of Algorithm SRCD for 100 iterations significantly decreases when the number of used threads increases in Fig. 4. In addition, the running time is acceptable for large applications. Hence, these results indicate that the proposed algorithm SRCD is feasible for large scale applications.

### REFERENCES


Duy Khuong Nguyen has received the B.E. and M.S. degrees in computer science from University of Engineering and Technology (UET), Vietnam National University Hanoi in 2008 and 2012, respectively. Currently, he is a Ph.D. student of joint doctoral program between Japan Advanced Institute of Science and Technology (JAIST), and University of Engineering and Technology (UET). His research interests include dimensionality reduction, algorithms, optimization and machine learning.

Tu Bao Ho is currently a professor of School of Knowledge Science, Japan Advanced Institute of Science and Technology. He received a B.T. in applied mathematics from Hanoi University of Science and Technology (1978), M.S. and Ph.D. in computer science from Pierre and Marie Curie University, Paris in 1984 and 1987. His research interests include knowledge-based systems, machine learning, knowledge discovery and data mining.