Multifrontal Non-negative Matrix Factorization

Piyush Sao[1] and Ramakrishnan Kannan[1]

Oak Ridge National Laboratory, {saopk,kannanr}@ornl.gov **

Abstract. Non-negative matrix factorization(NMF) is an important tool in high-performance large scale data analytics with applications ranging from community detection, recommender system, feature detection and linear and non-linear unmixing. While traditional NMF works well when the data set is relatively dense, however, it may not extract sufficient structure when the data is extremely sparse. Specifically, traditional NMF fails to exploit the structured sparsity of the large and sparse data sets resulting in dense factors. We propose a new algorithm for performing NMF on sparse data that we call multi-frontal NMF (MF-NMF) since it borrows several ideas from the multi-frontal method for unconstrained factorization (e.g. LU and QR). We also present an efficient shared memory parallel implementation of MF-NMF and discuss its performance and scalability. We conduct several experiments on synthetic and realworld datasets and demonstrate the usefulness of the algorithm by comparing it against standard baselines. We obtain a speedup of 1.2x to 19.5x on 24 cores with an average speed up of 10.3x across all the real world datasets.

Keywords: sparse matrix computations, distributed-memory parallelism, communication-avoiding algorithms

1 Introduction

Non-negative Matrix Factorization(NMF) is the problem of determining two non-negative matrices $W \in \mathbb{R}_+^{m \times k}$ and $H \in \mathbb{R}_+^{k \times n}$ such that $A \approx WH$ for the given matrix $A \in \mathbb{R}^{m \times n}$ with m samples and n features. Formally,

$$\text{argmin} W \geqslant 0, H \geqslant 0 \|A - WH\|_F^2 \tag{1}$$

where $\|X\|_F^2 = \sum_i \sum_j x_{ij}^2$.

** This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid- up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).This research was partially funded through Lab Director's Research and Development. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

With the advent of large scale internet data and interest in Big Data, researchers have started studying scalability of many foundational Data Mining and Machine Learning(DM/ML) algorithms. In the typical case of sparse matrices from internet data sets such as webgraph, bag of words matrices, $k \ll \min(m,n)$; for problems today, m and n can be on the order of millions or more, and k is on the order of few tens to thousands.

NMF is widely used in DM/ML as a dimension reduction method and for many real world problems as the non-negativity is inherent in many representations of real-world data and the resulting low rank factors are expected to have a natural interpretation. The applications of NMF range from text mining, computer vision and bioinformatics to blind source separation, unsupervised clustering and many other areas.

There is a vast literature on algorithms for NMF and their convergence properties [12]. The commonly adopted NMF algorithms are – (i) Multiplicative Update (MU) [17] (ii) Hierarchical Alternating Least Squares (HALS) [1,10] (iii) NMF based on Alternating Nonnegative Least Squares and Block Principal Pivoting (ABPP) [13], and (iv) Stochastic Gradient Descent (SGD) Updates [5]. As in Algorithm 1, most of the algorithms in NMF literature are based on alternately optimizing each of the low rank factors W and H while keeping the other fixed, in which case each subproblem is a constrained convex optimization problem. In this paper, we are considering HALS for explaining our proposed algorithm and experiments. But without loss of generality, it can be easily extended for other NMF algorithms as well.

It is trivial to understand that to approximate the sparse input matrix A, we need sparse W and H. To promote sparsity in the factors W and H [14], the above Equation (1), is extended with sparse ℓ_1 constraints as

$$\text{argmin} W \geqslant 0, H \geqslant 0 \|A - WH\|_F^2 + \|W\|_1 + \|H\|_1 \tag{2}$$

where $\|X\|_1 = \sum_i \sum_j abs(x_{ij})$.

Even though the factors from Equation (2) yields sparser factors over Equation (1), still the approximation error will be high, i.e in the order of 0.8 and 0.9. This is because, the low rank approximation WH will have non-zero entries in the place of zeros on the input matrix A, resulting in huge error. That is $\text{nnz}(WH) >> \text{nnz}(A)$. As a side effect, sparse NMF algorithm takes longer to converge.

To overcome this problem, we are proposing a Multifrontal Non-negative Matrix Factorization(MF-NMF) algorithm. Specifically, traditional NMF fails to exploit the structured sparsity of the large and sparse data sets. We propose a new algorithm for performing NMF on sparse data that we call multi-frontal NMF (MF-NMF) since it borrows several ideas from the multi-frontal method for unconstrained factorization (e.g. LU and QR). We show from Figs. 5 and 6 that the MF-NMF algorithm achieves 1.2x to 19.5x speed up with an average speedup of 10.3x on 24 cores without compromise in accuracy.

2 Background

In this section, we introduce the baseline NMF algorithm based on Hierarchical Alternative Least Squares(HALS) and explain the problems of the NMF algorithm on sparse data.

2.1 Non-negative Matrix Factorization(NMF)

The NMF [11] algorithms alternate between updating one of W and H using the given input matrix A and other 'fixed' factor - H for updating W or W for updating H. We show the structure of any NMF algorithm in Algorithm 1.

Algorithm 1 $[W,H]=\text{NMF}(A,k)$

Require: A is an $m\times n$ matrix, low rank k

1: Initialize H with a non-negative matrix in $\mathbb{R}_{+}^{n\times k}$.
2: **while** stopping criteria not satisfied **do**
3: Update W as $\text{argmin}_{\tilde{W}\geqslant 0}\left\|A-\tilde{W}H\right\|_F$
4: Update H as $\text{argmin}_{\tilde{H}\geqslant 0}\left\|A-W\tilde{H}\right\|_F$

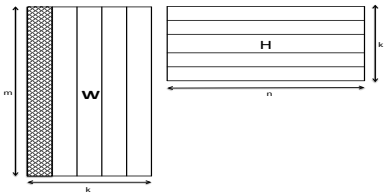


Fig. 1: HALS Algorithms that determines $2k$ vectors of W and H

The NMF algorithms vary in the realization of the steps 3 and 4 of Algorithm 1. In this paper, we are using a specific NMF algorithm called Hierarchical Alternating Least Squares (HALS) [1] that has the following update rule for solving Eq. (2)

$$\begin{aligned} H^i &\leftarrow [H^i + W^T A)^i - (W^T W + 2\beta \mathbf{1}_k)_i H]_+ \\ W_i &\leftarrow [(HH^T + 2\beta \mathbf{1}_k)_{ii} W_i + (AH^T)_i - W(HH^T + 2\beta \mathbf{1}_k)_i]_+ \end{aligned} \tag{3}$$

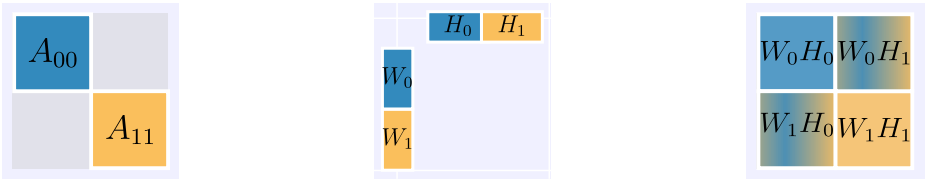
where $\mathbf{1}_k$ is a matrix of $k\times k$ with all one's, H^i is row vector, W_i is column vector and α,β are some positive scalars.

The convergence properties of different NMF algorithms are discussed in detail by Kim, He and Park [12]. While we focus only on the HALS algorithm in this paper, we highlight that our algorithm is not restricted to this, and is seamlessly extensible to other NMF algorithms as well, including HALS, ABPP, Alternating Direction Method of Multipliers (**ADMM**) [18], and Nesterov-based methods [8].

It can be shown that the Eq. (3) results in a relatively dense W and H for low rank k. Also, we can observe from Eq. (3), that there is no inherent parallelism on HALS algorithm other than the BLAS level parallelism available for sparse-dense matrix multiplication.

2.2 NMF on Sparse Datasets

In the baseline NMF algorithm Algorithm 1, the factor matrix W and H are dense even when A matrix is sparse. Thus, the approximant matrix $\hat{A}=WH$ is dense. This is undesirable in the case of structured sparse matrix since dense approximant $\hat{A}$ will not preserve the patterned sparsity of the matrix A such as the topic-modeling on bag of words representation and sparse adjacency matrix representation of large scale social network graphs.



(a) Block Diagonal Input Matrix (b) Factor Matrices (c) Approximation

Fig. 2: Dense Factors Problem on Illustrative Block Diagonal Sparse NMF

Consider the NMF of 2×2 sparse block diagonal matrix as shown in Fig. 2a. This block-diagonal structure of the matrix suggests that the data is *separable* into two disjoint sets described by matrices A_{00} and A_{11}. When we perform NMF of this matrix, we obtain two dense factors $W=\begin{bmatrix}W_0\\W_1\end{bmatrix}$ and $H=\begin{bmatrix}H_0 & H_1\end{bmatrix}$ (Fig. 2b). Hence the non-negative approximation of the matrix A is dense (Fig. 2c). Thus this approximation loses the separability property of the input matrix.

Furthermore, the error of non-negative approximation for the 2×2 sparse block diagonal matrix is given by

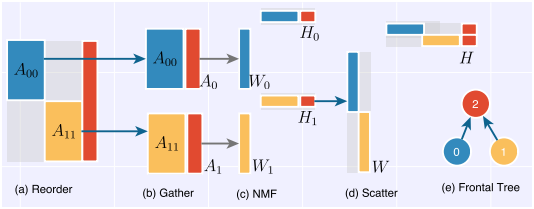
$$\|A-WH\|_F^2 \approx \|A_{00}-W_0H_0\|_F^2+\|A_{11}-W_1H_1\|_F^2+\|W_1H_0\|_F^2+\|W_0H_1\|_F^2.$$

Note that the first two terms $\|A_{00}-W_0H_0\|_F^2$ and $\|A_{11}-W_1H_1\|_F^2$ are more than the approximation error when we perform NMF on blocks A_{00} and A_{11}, respectively. Additionally, $\|W_1H_0\|_F^2$ $\|W_0H_1\|_F^2$ are spurious and non-zero and they further worsen the approximation error.

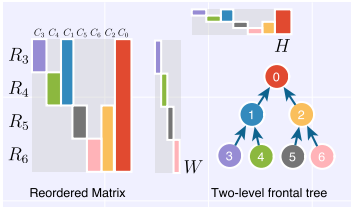
We summarize the drawbacks of performing Algorithm 1 to compute NMF of a sparse matrix as (a) W and H factors are dense (b) NMF approximation does not preserve the structural properties of the data and (c) the error of approximation (and thus the approximation quality) is not optimal.

Informal Problem Statement: To overcome the limitations of Algorithm 1, we seek an NMF algorithm that preserve the structural sparsity of the input data in the approximation $\hat{A}=WH$. Since the factor matrices are non-negative, no numerical cancellation can occur. Hence if the approximation $\hat{A}$ is sparse then W and H will also be sparse.

3 Multifrontal Non-negative Matrix Factorization



(a) MF-NMF on 2×3 block sparse matrix



(b) MF-NMF factors with two level ND

Fig. 3: On Fig. 3a, we show the four steps of MF-NMF and the frontal tree for one level-ND. Figure 3a shows reordered matrix, W and H factors and frontal tree for two level-ND.

First, we describe the working of MF-NMF on a 2×3 block-sparse matrix shown in Fig. 3a-a. In MF-NMF, we reorder the input matrix A to expose the sparse block structure as shown inFig. 3a-a and instead of performing rank-k NMF on the whole matrix A, we divide it into two smaller sub-problems $A_0 = \begin{bmatrix} A_{00} & A_{02} \end{bmatrix}$ and $A_1 = \begin{bmatrix} A_{10} & A_{12} \end{bmatrix}$, and perform a $k/2$ rank NMF on $A_0 \approx_+ W_0 H_0$ and $A_1 \approx_+ W_1 H_1$. Finally, W_0 and W_1, and H_0 and H_1, are scattered to form the final W and H factors. We perform this process in the following four steps (Fig. 3a).

1. Reorder: we compute a column permutation P, and reorder the matrix to obtain a block sparse matrix structure as shown in Fig. 3a.
2. Gather: We gather the sparse blocks into multiple smaller subproblems A_i.
3. NMF: We perform independent NMF on each $A_i \approx_+ W_i H_i$ using Algorithm 1.
4. Scatter: We scatter all W_i and H_i to construct the final factor matrices W and H.

Algorithm 2 $[W,H] = \text{MF-NMF}(A,k,h)$

Require: A is an $m \times n$ matrix, low rank k, height h of frontal tree

1: $F,P \leftarrow$ Reorder (A,h) % *F is frontal tree and P is permutation(Section 3.1)*
2: $\ell \leftarrow \#\text{leaf}(F)$
3: $A_0, A_1, \cdots, A_\ell \leftarrow$ Gather (A,F,P)
4: **for** $i = 1$ to ℓ **do**
5: $\quad [W_i, H_i] \leftarrow$ NMF $(A_i, \frac{k}{\ell})$
6: $W,H \leftarrow$ Scatter $(\begin{bmatrix} W_1, H_1 \end{bmatrix}, \begin{bmatrix} W_2, H_2 \end{bmatrix}, \cdots \begin{bmatrix} W_\ell, H_\ell \end{bmatrix})$

The only non-trivial step in MF-NMF is reordering. In this paper, we use the so nested-dissection (ND) based graph-partition to obtain desired ordering.

3.1 Nested-Dissection(ND) Based Matrix Reordering

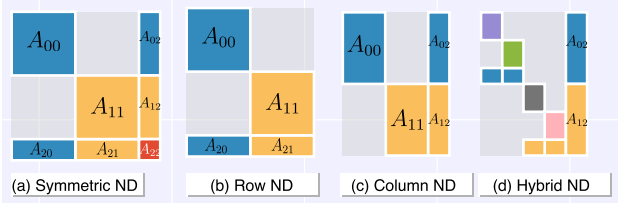


Fig. 4: Different ND reorderings. In the first order shows a symmetric reordering, and the second and third case show row and column ND respectively. In the last case, we do a column-ND at the top level; and perform a row-ND for A_{00} and A_{11}.

Without loss of generality, we assume that the input sparse matrix A is a tall rectangular matrix. Typical ND is performed on a symmetric sparse matrix. So we first describe ND for the symmetric case and then we show how do we use symmetric ND for ordering rectangular sparse matrices.

Symmetric ND For a symmetric sparse matrix $A \in \mathbb{R}^{n \times n}$, there is a corresponding undirected graph $G=(V,E)$ with $|V|=n$ vertices corresponding to each column in A; and $|E|=nnz(A)$ edges, where any edge e_{ij} corresponds to a non-zero entry A_{ij}.

In ND of the matrix A, we find a vertex separator $S \subset V$, that partitions the $V=C_1 \cup S \cup C_2$, such that there are no edges between any vertex in C_1 to any vertex in C_2. Using this partition, we reorder the matrix so that a) vertices in each set C_1, C_2 and S numbered consecutively; and b) vertices in S are numbered at the end. In the symmetric case, the matrix is reordered symmetrically, i.e., if P is the permutation matrix, then reordered matrix $\hat{A}=P^T AP$, so $\hat{A}$ is also symmetric.

We can obtain such a permutation using graph partitioning tools such as Metis and Scotch. The re-ordered matrix has an *arrow-head* structure as shown in Fig. 4-a. In Fig. 4-a, A_{00}, A_{11} and A_{22} correspond to C_1, C_2 and S,respectively.

Unsymmetric ND: When A is a rectangular matrix, we can perform ND on $B_c=A^T A$ or $B_r=AA^T$. First consider $B_c=A^T A$. Let P_c be the permutation matrix obtained by performing ND on $B_c=A^T A$. Using P_c, we permute the columns of the matrix A to obtain $\hat{A}_c=AP_c$. We call this reordering scheme as column-ND. $\hat{A}_c$ has a rotated-staircase structure as shown in Fig. 4-c. Similarly, we can perform ND on B_r to obtain a permutation matrix P_r and permute the matrix A to obtain $\hat{A}_r=P_r A$, which has a reverse-staircase structure as shown in Fig. 4-b. We call this reordering scheme as row-ND.

In general, the approximation quality $\frac{||A-WH||_F}{||A||_F}$ of factors can vary significantly with the choice of reordering scheme, and it depends on the nature of data itself. Informally, the reordering scheme that best captures the natural structure of the data, will result in best approximation quality. One may use more complex reordering scheme such as alternating between column-ND and row-ND. For example, in Fig. 4-d

we use column-ND in the first step and us row-ND for partitioning A_{00} and A_{11}. In this paper, we keep our discussion limited to column-ND.

3.2 Gather substep

In the Gather step, we gather smaller block sparse matrices to form disjoint submatrices A_i that we call frontal matrix. To do so, we need to know the number of frontal matrices A_i, and its row and column set. The number of submatrices depends on the *depth* of ND performed in the reordered step. If we perform ND only once as in the case of Fig. 3a, we get two submatrices and for ND of depth two in Fig. 3b we get four submatrices.

Frontal Tree: The column structure of frontal can be expressed conveniently using a tree, that we call *frontal-tree*, which is analogous to the so called-elimination tree of the sparse Cholesky factorization. The frontal tree for ND of depth one and two are shown in Fig. 3a and Fig. 3b, respectively. Each node i in the frontal tree corresponds to a subset of column C_i. The number of frontal matrices is the number of leaf nodes in the frontal tree.

The row set R_i^F of a frontal matrix A_i corresponding to a leaf node i is the set of non-zero rows in $A_{:,C_i}$ where C_i is the set of columns corresponding to i-th node in the frontal tree. The column set C_i^F of a frontal matrix A_i is the union of columns in all the ancestors of the leaf node i in the frontal tree and itself. For example, in Fig. 3b, the leaf node 3 has ancestors 1 and 0 so the frontal matrix corresponding to will have the $C_3^F = C_3 \cup C_1 \cup C_0$ as the column set. Using R_i^F and C_i^F, the i-th frontal matrix can be obtained as $A_i = A_{R_i^F, C_i^F}$.

After the Gather step, we perform independent NMF on all the frontal matrix and, if required, in Scatter the factors to form the final W and H matrices.

The complete algorithm appears in Algorithm 2. We use the desired height of the frontal tree, h as a user input, which controls the number of frontal matrices. Note that the number of frontal matrices can be at most k, where k is the rank of the final non-negative factors.

4 Experiments and Evaluation

In this section, we present results from a series of numerical experiment to understand the scalability of 3D sparse triangular solver algorithm.

4.1 Experimental Set-up

The entire experiment was conducted on a node in Rhea commodity-type Linux cluster at Oak Leadership Computing Facility. Every node has two Intel® Xeon® E5-2650 at 2.0 GHz with each having 8 physical cores and 16 Hyperthreads with a total of 16 cores and 32 hyperthreads per node. We compile the code with gcc 6.2.0 compiler with "-fopenmp" flag for openmp threads.

4.2 Datasets

For the synthetic experiments, we generated standard normal random matrices and the details of the real world datasets are presented in Table 1.

We use a mix of matrices from real world and scientific applications. The matrices *psse0 and psse1* are related to powergrid and the rest of the matrices belong belong to scientific applications such as combinatorial, linear programming and quantum chemistry.

Both the real world and the synthetic matrices are shifted with the minimum value such that there is no negative elements in the input matrix.

Table 1: Test sparse matrices used in experiments

Name	Source	m	n	$\frac{nnz}{\max(m,n)}$	Name	Source	m	n	$\frac{nnz}{\max(m,n)}$
Franz11	Comb	4.7e4	3.0e4	7	Franz5	Comb	7.3e3	2.8e3	5.9
Franz7	Comb	1.0e4	1.7e3	4.1	Catears24	Comb	1.0e3	2.6e3	2.9
kneser831	Comb	1.5e4	1.5e4	2.9	mk10-b2	Comb	3.1e3	6.3e2	3
rosen2	Linear Prog	1.0e3	3.0e3	15.4	SiNa	Quantum Chemistry	5.7e3	5.7e3	34.61
psse0	Power	2.6e4	1.1e4	3.83	psse1	Power	1.4e4	1.1e4	4.0

4.3 Metrics

We are reporting relative error as a quality metric and per iteration time as performance metric.

Relative Error: Relative error defines the approximation of the obtained factor matrices W and H against the input matrix as defined below.

$$\text{Relative Error} = \frac{\|A - WH\|_F^2}{\|A\|_F^2} \tag{4}$$

Average Per Iteration Time in Seconds: NMF algorithms are iterative in nature. There are many stopping criterion and the most common among them are (a) difference of relative error between two successive iterations (b) tolerance on the relative error and (c) number of iterations. In this paper, we are specifically using HALS NMF algorithm and stopping after 20 iterations. All the baselines and the proposed algorithm were run for 20 iterations. For most of the datasets, it is common in the community to run for 20-30 iterations, some datasets might need longer. Hence, we are reporting the average per iteration time in seconds as the performance metric.

For both the time and the error, we also report the relative performance by normalizing with the minimum value on every dataset. That is., the in both Fig. 6 and Fig. 5 for *franz11*, we normalized every accuracy and the time with the minimum from that experiment.

4.4 Real World Data

Parallel Performance As in Fig. 5, we obtained a speedup of 1.2x to 19.5x on *mk10b2* with an average speed up of 10.3x across all the real world datasets. The level of parallelism and the speedup is directly proportional to the height of the tree that is dependent on the sparsity pattern. For realworld data with good sparsity pattern like *psse0* and *Franz*, we are able to obtain trees with height upto level 7. Where as, *cat-ears* and *rosen2*, cannot obtain more than two levels. In the case of *Franz7, Franz11*, at height 7, the speedup is 16x and 10x for *psse*. Sometimes it is difficult to obtain balanced partition for a given level – that is., equal number of nnz's across partition; hurting the speedup. For example, in *mk10b2*, at level 1, with two leaf nodes, we obtain a speedup of only 1.2x which is alleviated in higher levels achieving a peak speedup of 19.5x. Overall, the proposed Algorithm 2, gave good speedup for sufficiently sparse matrices with good sparsity pattern.

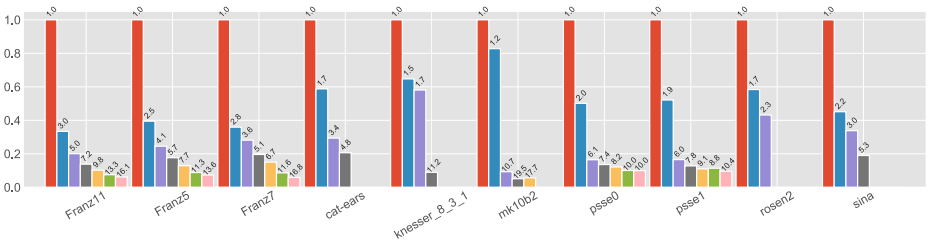


Fig. 5: Parallel Performance of MF-NMF on real world datasets by varying the height of frontal tree. Every bar shows the relative speedup with respect to the baseline Algorithm 1

Accuracy We are reporting the relative accuracy with the global matrix. That is., we compute the relative error of the proposed algorithm Algorithm 2 and normalize with the relative error of global matrix. Every red bar in Fig. 6 is for the global matrix with relative accuracy always 1.0 and all the rest are normalized with this absolute value of the relative error. The relative accuracy of Algorithm 2 is in between 1.0 to 1.04 – that is., in a worst case there was only 4% deviation from the global relative error. The absolute relative error for global matrices for *Franz11,Franz5,Franz7, cat-ears, knesser, mk10b2, psse0, psse1, rosen2* and *sina* are 0.998246, 0.971278, 0.97998, 0.97805, 0.996353, 0.988146, 0.934138, 0.893648, 0.904814 and 0.998638 respectively.

5 Related Work

Our proposed MF-NMF borrows heavily from direct methods for sparse linear system of equations which goes back to seminal work George [6]. Comprehensive discussion of sparse LU and Cholesky factorization, including matrix reordering and the elimination

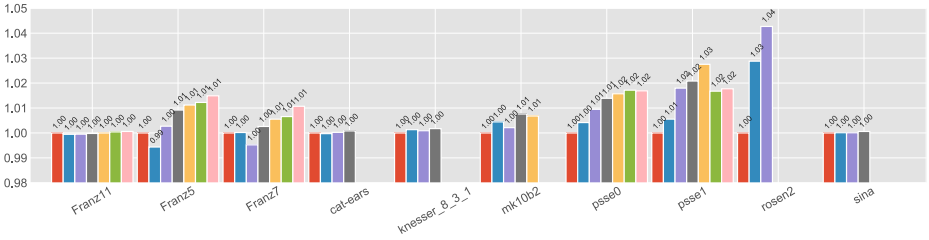


Fig. 6: Relative accuracy of MF-NMF with respect to the baseline Algorithm 1 by varying the height of the frontal tree

tree can be found in [3,9]. Matrix reordering used for MF-NMF is more closely related to sparse QR factorization [2] and unsymmetric LU factorization[7].

The popular algorithm for NMF is Multiplicative Update(MU) and there are literature that focuses on distributed implementation of MU on Hadoop[15,19,16] and Spark. These explored Matrix multiplication, element-wise multiplication, and element-wise division are the building blocks of the MU algorithm.

While we do not discuss parallelism aspect of MF-NMF, design of MF-NMF is also motivated by improving the parallelism in NMF. Sparse NMF algorithms on shared memory environment are heavily reliant on parallelism available on BLAS/LAPACK libraries for Sparse-Dense matrix operations [4]. For the first time in the literature, the proposed MF-NMF exposes more parallelism in shared memory environment by exploiting the sparsity of the data set.

6 Conclusion

The traditional NMF fails to exploit the structured sparsity of the large and sparse data sets. In this paper we proposed a MF-NMF algorithm that uses concepts from sparse direct methods and graph partitioning to exploit the structured sparsity. Our initial assessment suggests MF-NMF can be much faster than traditional NMF while incurring negligible penalty in terms of accuracy. Our reordering scheme based on ND, is only a first step towards exploiting sparsity for NMF and we believe exploring more sophisticated reordering schemes based on domain knowledge is warrented. In the future, we would like to extend the work with (a) the distributed communication avoiding variant and (b) quantify the usefulness of the factors obtained from the proposed algorithm on internet scale data.

References

1. Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. Wiley, 2009.
2. Timothy Davis. Multifrontral multithreaded rank-revealing sparse qr factorization. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

3. Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Clarendon press Oxford, 1986.
4. James P. Fairbanks, Ramakrishnan Kannan, Haesun Park, and David A. Bader. Behavioral clusters in dynamic graphs. *Parallel Computing*, 47:38–50, 2015.
5. Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the KDD*, pages 69–77. ACM, 2011.
6. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.
7. Laura Grigori, Erik G Boman, Simplice Donfack, and Timothy A Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization. *SIAM Journal on Scientific Computing*, 32(6):3426–3446, 2010.
8. N. Guan, D. Tao, Z. Luo, and B. Yuan. Nenmf: An optimal gradient method for nonnegative matrix factorization. *IEEE Transactions on Signal Processing*, 60(6):2882–2898, June 2012.
9. Michael T Heath, Esmond Ng, and Barry W Peyton. Parallel algorithms for sparse linear systems. *SIAM review*, 33(3):420–460, 1991.
10. Ngoc-Diep Ho, Paul Van Dooren, and Vincent D. Blondel. Descent methods for nonnegative matrix factorization. *CoRR*, abs/0801.3199, 2008.
11. Ramakrishnan Kannan, Grey Ballard, and Haesun Park. Mpi-faun: an mpi-based framework for alternating-updating nonnegative matrix factorization. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):544–558, 2018.
12. Jingu Kim, Yunlong He, and Haesun Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.
13. Jingu Kim and Haesun Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281, 2011.
14. Wooyoung Kim, Bernard Chen, Jingu Kim, Yi Pan, and Haesun Park. Sparse nonnegative matrix factorization for protein sequence motif discovery. *Expert Syst. Appl.*, 38(10):13198–13207, 2011.
15. Ruiqi Liao, Yifan Zhang, Jihong Guan, and Shuigeng Zhou. CloudNMF: A MapReduce implementation of nonnegative matrix factorization for large-scale biological datasets. *Genomics, proteomics & bioinformatics*, 12(1):48–51, 2014.
16. Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on MapReduce. In *Proceedings of the WWW*, pages 681–690. ACM, 2010.
17. D. Seung and L. Lee. Algorithms for non-negative matrix factorization. *NIPS*, 13:556–562, 2001.
18. D. L. Sun and C. Févotte. Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6201–6205, May 2014.
19. Jiangtao Yin, Lixin Gao, and Zhongfei(Mark) Zhang. Scalable nonnegative matrix factorization with block-wise updates. In *Machine Learning and Knowledge Discovery in Databases*, volume 8726 of *LNCS*, pages 337–352, 2014.