Noname manuscript No.
(will be inserted by the editor)

S^3CMTF: Fast, Accurate, and Scalable Method for Sparse Coupled Matrix-Tensor Factorization

Dongjin Choi · Jun-Gi Jang · U Kang

Received: date / Accepted: date

Abstract How can we extract hidden relations from a tensor and a matrix data simultaneously in a fast, accurate, and scalable way? Coupled matrix-tensor factorization (CMTF) is an important tool for the purpose. Designing an accurate and efficient CMTF method has become more crucial as the size and dimension of real-world data are growing explosively. However, existing methods for CMTF suffer from lack of accuracy, slow running time, and limited scalability.

In this paper, we propose S^3CMTF, a fast, accurate, and scalable CMTF method. In contrast to previous methods which do not support sparse tensors or do not model complicated relationships between factors, S^3CMTF provides sparse Tucker factorization by carefully deriving gradient update rules. We also show that lock-free parallel SGD is useful for S^3CMTF in multi-core shared memory systems. S^3CMTF further boosts the performance by carefully storing intermediate computation and reusing them. We theoretically and empirically show that S^3CMTF is the fastest, outperforming existing methods. Experimental results show that S^3CMTF is up to 989× faster than existing methods while providing the best accuracy. S^3CMTF shows linear scalability on the number of data entries and the number of cores. In addition, we apply S^3CMTF to Yelp recommendation tensor data coupled with 3 additional matrices to discover interesting patterns.

Keywords Sparse coupled matrix-tensor factorization · Tucker decomposition · Recommendation system

1 Introduction

Given a tensor data, and related matrix data, how can we analyze them efficiently? Tensors (i.e., multi-dimensional arrays) and matrices are natural representations

Dongjin Choi
E-mail: skywalker5@snu.ac.kr

Jun-Gi Jang
E-mail: elnino9158@gmail.com

U Kang
E-mail: ukang@snu.ac.kr

Table 1: Comparison of our proposed S^3CMTF and the existing CMTF methods. S^3CMTF outperforms all other methods in terms of time, accuracy, scalability, memory usage, and parallelizability.

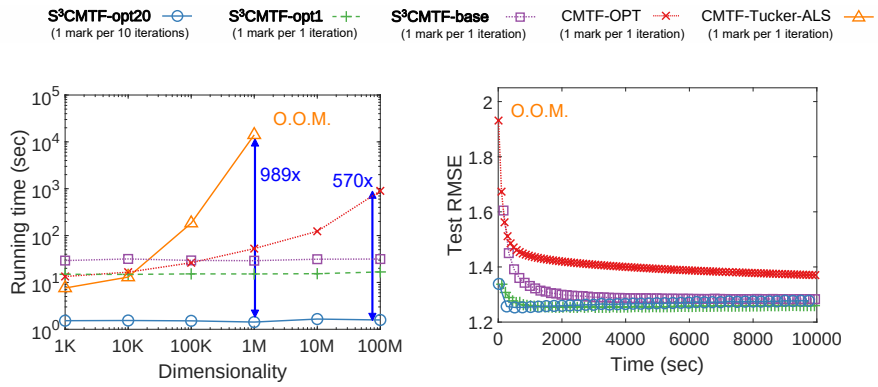
Method	**Time**	**Accuracy**	**Scalability**	**Memory**	**Parallel**
CMTF-Tucker-ALS	slow	low	low	high	no
CMTF-OPT	slow	low	low	high	no
S^{3}CMTF-base	fast	**high**	**high**	**lower**	**yes**
S^{3}CMTF-opt	**faster**	**high**	**high**	low	**yes**

for various real world high-order data. For instance, an online review site Yelp provides rich information about users (name, friends, reviews, etc.), or businesses (name, city, Wi-Fi, etc.). One popular representation of such data includes a 3-way rating tensor with (user ID, business ID, time) triplets and an additional friendship matrix with (user ID, user ID) pairs. Coupled matrix-tensor factorization (CMTF) is an effective tool for joint analysis of coupled matrices and a tensor. The main purpose of CMTF is to integrate matrix factorization [17] and tensor factorization [15] to efficiently extract the factor matrices of each mode. The extracted factors have many useful applications such as latent semantic analysis [7, 23, 28], recommendation systems [12, 25], network traffic analysis [26], and completion of missing values [1, 2, 19].

However, existing CMTF methods do not provide good performance in terms of time, accuracy, and scalability. CMTF-Tucker-ALS [21], a method based on Tucker decomposition [6], has a limitation that it is only applicable for dense data. For sparse real-world data, it assumes empty entries as zero and outputs highly skewed results which are impractical. Moreover, CMTF-Tucker-ALS does not scale to large data because it suffers from high memory requirement caused by *M-bottleneck problem* [20] (see Section 2.3 for details). CMTF-OPT [1] is a CMTF method based on CANDECOMP/PARAFAC (CP) decomposition [15]. It has a limitation that it does not take advantage of all inter-relations between related factors because CP model represents a limited case of the Tucker model in which each factor is related to only a few of other factors. Therefore, CMTF-OPT undergoes the low model capacity and results in high test error.

In this paper, we propose S^3CMTF (**S**parse, lock-free **S**GD based, and **S**calable CMTF), a fast, accurate, and scalable CMTF method which resolves the problems of previous methods. Unlike previous methods which do not support sparse tensors or do not model complicated relationships between factors, S3CMTF provides sparse Tucker factorization by carefully deriving gradient update rules. We also show that lock-free parallel SGD is useful for S^3CMTF in multi-core shared memory systems. S^3CMTF further boosts the performance by carefully storing intermediate computation and reusing them. Table 1 shows the comparison of S^3CMTF and other existing methods. The main contributions of our study are as follows:

- **Algorithm:** We propose S^3CMTF, a fast, accurate, and scalable coupled tensor-matrix factorization algorithm for matrix-tensor joint datasets. S^3CMTF



(a) **Running time vs. dimensionality** (b) **Error vs. running time**

Fig. 1: Comparison of our proposed S^3CMTF and the existing methods. (a) For a fixed number of nonzeros, S^3CMTF takes constant time as dimensionality grows, while existing methods become slower. Our best method S^3CMTF-opt20 is 570× and 989× faster than existing methods. (b) S^3CMTF-opt20 shows the best convergence rate and accuracy on real world Yelp dataset. O.O.M.: out of memory error.

is designed to efficiently extract factors from the joint datasets by taking advantage of sparsity, exploiting intermediate data, and parallelization.
- **Performance:** S^3CMTF shows the best performance on accuracy, speed, and scalability. S^3CMTF runs up to **989× faster** and is more scalable than existing methods as shown in Figure 1a. For real-world datasets, S^3CMTF converges faster to the better optimum as shown in Figures 1b and 4.
- **Discovery:** Applying S^3CMTF on Yelp review dataset with a 3-mode tensor (user, business, time) coupled with 3 additional matrices ((user, user), (business, category), and (business, city)), we observe interesting patterns and clusters of businesses and suggest a process for personal recommendation.

The rest of paper is organized as follows. Section 2 gives the preliminaries and related works of the tensor and CMTF. Section 3 describes our proposed S^3CMTF method for fast, accurate and scalable CMTF. Section 4 shows the results of performance experiments for our proposed method. After presenting the discovery results in Section 5, we conclude in Section 6.

2 Preliminaries and Related Works

In this section, we describe preliminaries for tensor and coupled matrix-tensor factorization. We list all symbols used in this paper in Table 2.

2.1 Tensor

A tensor is a multi-dimensional array. Each 'dimension' of a tensor is called *mode* or *way*. The length of each mode is called 'dimensionality' and denoted by $I_1, \cdots, I_N$.

Table 2: Table of symbols.

Symbol	Definition
$\mathcal{X}$	input tensor
$\mathcal{G}$	core tensor
N	order (number of modes) of the input tensor
I_n	dimensionality of n-th mode of input tensor $\mathcal{X}$
J_n	dimensionality of n-th mode of core tensor $\mathcal{G}$
α	a tensor index $(i_1 i_2 \cdots i_N)$
x_α	the entry of $\mathcal{X}$ with index α
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor
$\mathbf{U}^{(n)}$	n-th factor matrix of $\mathcal{X}$
$\{\mathbf{U}\}$	set of all factor matrices of $\mathcal{X}$
$\mathbf{u}_i^{(n)}$	the i-th row vector of $\mathbf{U}^{(n)}$
$\{\mathbf{u}\}_\alpha$	ordered set of row vectors $\{\mathbf{u}_{i_1}^{(1)}, \mathbf{u}_{i_2}^{(2)}, \ldots, \mathbf{u}_{i_N}^{(N)}\}$
$\{\mathbf{u}\}_\alpha^\mathsf{T}$	ordered set of column vectors $\{\mathbf{u}_{i_1}^{(1)\mathsf{T}}, \mathbf{u}_{i_2}^{(2)\mathsf{T}}, \ldots, \mathbf{u}_{i_N}^{(N)\mathsf{T}}\}$
$u_{ij}^{(n)}$	entry of $\mathbf{U}^{(n)}$ with index (i, j)
$\mathbf{Y}$	coupled matrix
β	a matrix index $k_1 k_2$
y_β	the entry of $\mathbf{Y}$ with index β
$\mathbf{V}$	factor matrix for the coupled matrix $\mathbf{Y}$
$\mathbf{v}_k$	the k-th row vector of $\mathbf{V}$
$\Omega_\mathcal{X}$	index set of $\mathcal{X}$
$\Omega_\mathcal{X}^{n,i}$	subset of $\Omega_\mathcal{X}$ having i as the n-th index

In this paper, an N-mode or N-way tensor is denoted by the boldface Euler script capital (e.g. $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$), and matrices are denoted by boldface capitals (e.g. $\mathbf{A}$). x_α and a_β denote the entry of $\mathcal{X}$ and $\mathbf{A}$ with indices α and β, respectively.

We describe tensor operations used in this paper. A mode-n fiber is a vector which has fixed indices except for the n-th index in a tensor. The mode-n matrix product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{A}$ and has the size of $I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \cdots \times I_N$. It is defined as:

$$(\mathcal{X} \times_n \mathbf{A})_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \ldots i_N} a_{j i_n} \tag{1}$$

where $a_{j i_n}$ is the (j, i_n)-th entry of $\mathbf{A}$. For brevity, we use following shorthand notation for multiplication on every mode as in [16]:

$$\mathcal{X} \times \{\mathbf{A}\} := \mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} \tag{2}$$

where $\{\mathbf{A}\}$ denotes the ordered set $\{\mathbf{A}^{(\mathbf{1})}, \mathbf{A}^{(2)}, \cdots, , \mathbf{A}^{(N)}\}$.

We use the following notation for multiplication on every mode except n-th mode.

$$\mathcal{X} \times_{-n} \{\mathbf{A}\} := \mathcal{X} \times_1 \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}$$

We examine the case that an ordered set of row vectors $\{\mathbf{a}^{(\mathbf{1})}, \mathbf{a}^{(\mathbf{2})}, \cdots, \mathbf{a}^{(\mathbf{N})}\}$, denoted by $\{\mathbf{a}\}$, is multiplied to a tensor $\mathcal{X}$. First, consider the multiplication for

every corresponding mode. By Equation (1),

$$\mathcal{X} \times \{\mathbf{a}\} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)}$$

where $a_k^{(m)}$ denotes the k-th element of $\mathbf{a}^{(m)}$. Then, consider the multiplication for every mode except n-th mode. Such multiplication results to a vector of length I_n. The k-th entry of the vector is

$$\left[\mathcal{X} \times_{-n} \{\mathbf{a}\}\right]_k = \sum_{\forall \alpha \in \Omega_{\mathcal{X}}^{n,k}} x_\alpha a_{i_1}^{(1)} \cdots a_{i_{n-1}}^{(n-1)} a_{i_{n+1}}^{(n+1)} \cdots a_{i_N}^{(N)} \tag{3}$$

where $\Omega_{\mathcal{X}}^{n,k}$ denotes the index set of $\mathcal{X}$ having its n-th index as k. $\alpha = (i_1 i_2 \cdots i_N)$ denotes the index for an entry.

2.2 Tucker Decomposition

Tucker decomposition is one of the most popular tensor factorization models. Tucker decomposition approximates an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ and factor matrices $\mathbf{U}^{(1)} \in \mathbb{R}^{I_1 \times J_1}, \mathbf{U}^{(2)} \in \mathbb{R}^{I_2 \times J_2}, \ldots, \mathbf{U}^{(N)} \in \mathbb{R}^{I_N \times J_N}$ satisfying

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)} = \mathcal{G} \times \{\mathbf{U}\}$$

Element-wise formulation of Tucker model is

$$\begin{aligned} x_\alpha \approx \tilde{x}_\alpha &= \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} g_{j_1 j_2 \cdots j_N} u_{i_1 j_1}^{(1)} u_{i_2 j_2}^{(2)} \cdots u_{i_N j_N}^{(N)} \\ &= \mathcal{G} \times_1 \mathbf{u}_{i_1}^{(1)} \times_2 \mathbf{u}_{i_2}^{(2)} \cdots \times_N \mathbf{u}_{i_N}^{(N)} := \mathcal{G} \times \{\mathbf{u}\}_\alpha \end{aligned} \tag{4}$$

where α is a tensor index $(i_1 i_2 \cdots i_N)$, and $\mathbf{u}_{i_n}^{(n)}$ denotes the i_n-th row of factor matrix $\mathbf{U}^{(n)}$. $\{\mathbf{u}\}_\alpha$ denotes the set of factor rows $\{\mathbf{u}_{i_1}^{(1)}, \mathbf{u}_{i_2}^{(2)}, \cdots, \mathbf{u}_{i_N}^{(N)}\}$. Note that the core tensor $\mathcal{G}$ implies the relation between the factors in Tucker formulation. When the core tensor size satisfies $J_1 = J_2 = \cdots = J_N$ and the core tensor $\mathcal{G}$ is hyper-diagonal, it is equivalent to CANDECOMP/PARAFAC (CP) decomposition. There is orthogonality constraint for Tucker decomposition: each factor matrix is a column-wise orthogonal matrix (e.g. $\mathbf{U}^{(n)T}\mathbf{U}^{(n)} = \mathbf{I}$ for $n = 1, \cdots, N$ where $\mathbf{I}$ is an identity matrix).

2.3 Coupled Matrix-Tensor Factorization

Coupled matrix-tensor factorization (CMTF) is proposed for collective factorization of a tensor and matrices. CMTF integrates matrix factorization and tensor factorization.

Definition 1 (Coupled Matrix-Tensor Factorization) Given an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and a matrix $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$ where c is the coupled mode, $\mathcal{X} \approx \tilde{\mathcal{X}} = \mathcal{G} \times \{\mathbf{U}\}$, $\mathbf{Y} \approx \tilde{\mathbf{Y}} = \mathbf{U}^{(c)}\mathbf{V}^{\mathsf{T}}$ are the coupled matrix-tensor factorization. $\mathbf{U}^{(c)} \in \mathbb{R}^{I_c \times J_c}$ is the c-th mode factor matrix, and $\mathbf{V} \in \mathbb{R}^{K \times J_c}$ denotes the factor matrix for coupled matrix. Finding the factor matrices and core tensor for CMTF is equivalent to solving

$$\underset{\mathbf{U}^{(1)},\cdots,\mathbf{U}^{(N)},\mathbf{V},\mathcal{G}}{\arg\min} \|\mathcal{X} - \mathcal{G} \times \{\mathbf{U}\}\|^2 + \|\mathbf{Y} - \mathbf{U}^{(c)}\mathbf{V}^{\mathsf{T}}\|^2 \tag{5}$$

where $\| \bullet \|$ denotes the Frobenius norm.

Various methods have been proposed to efficiently solve the CMTF problem. An alternating least squares (ALS) method CMTF-Tucker-ALS [21] is proposed. CMTF-Tucker-ALS is based on Tucker-ALS (HOOI) [6] which is a popular method for solving Tucker model. Tucker-ALS suffers from a crucial intermediate memory-bottleneck problem known as *M-bottleneck problem* [20] that arises from materialization of a large dense tensor $\mathcal{X} \times_{-n} \{\mathbf{U}\}^{\mathsf{T}}$ as intermediate data where $\{\mathbf{U}\}^{\mathsf{T}} = \{\mathbf{U}^{(1)\mathsf{T}}, \mathbf{U}^{(2)\mathsf{T}}, \cdots, \mathbf{U}^{(N)\mathsf{T}}\}$.

Most existing methods use CP decomposition model for $\tilde{\mathcal{X}}$ where $J_1 = J_2 = \cdots = J_N$ and the core tensor $\mathcal{G}$ is hyper-diagonal [1, 9, 11, 22, 3]. CMTF-OPT [1] is a representative algorithm for CMTF using CP decomposition model which uses gradient descent method to find factors. HaTen2 [11, 10], and SCouT [9] propose distributed methods for CMTF using CP decomposition model. Turbo-SMT [22] provides a time-boosting technique for CP-based CMTF methods.

Note that Equation (5) requires entire data entries of $\mathcal{X}$ and $\mathbf{Y}$. It shows low accuracy when $\mathcal{X}$ and $\mathbf{Y}$ are sparse since empty entries are set to zeros even when they are irrelevant. For example, an empty entry in movie rating data does not mean score 0. For the reason above methods show low accuracy for real-world sparse data; what we focus on this paper is solving CMTF for sparse data.

Definition 2 (Sparse CMTF) When $\mathcal{X}$ and $\mathbf{Y}$ are sparse, sparse CMTF aims to find factors only considering observed entries. Let $\mathcal{W}^{(1)}$ and $\mathbf{W}^{(2)}$ indicate the observed entries of $\mathcal{X}$ and $\mathbf{Y}$ such that

$$w_\alpha^{(1)}\left(w_\beta^{(2)}\right) = \begin{cases} 1 & \text{if } x_\alpha\left(y_\beta\right) \text{ is known} \\ 0 & \text{if } x_\alpha\left(y_\beta\right) \text{ is missing} \end{cases}, \text{ for } \forall\alpha \in \Omega_{\mathcal{X}}\left(\forall\beta \in \Omega_{\mathbf{Y}}\right)$$

We modify Equation (5) as

$$\underset{\mathbf{U}^{(1)},\cdots,\mathbf{U}^{(N)},\mathbf{V},\mathcal{G}}{\arg\min} \|\mathcal{W}^{(1)} * (\mathcal{X} - \mathcal{G} \times \{\mathbf{U}\})\|^2 + \|\mathbf{W}^{(2)} * (\mathbf{Y} - \mathbf{U}^{(c)}\mathbf{V}^{\mathsf{T}})\|^2 \tag{6}$$

where $*$ denotes the Hadamard product (element-wise product).

CMTF-Tucker-ALS does not support sparse CMTF. For CP model, CMTF-OPT provides single machine approach for sparse CMTF, and CDTF [27] and FlexiFaCT [3] provide distributed methods for sparse CMTF. However, CP model suffers from high error because it does not capture the correlations between different factors of different modes because its core tensor has only hyper-diagonal nonzero entries [13].

3 Proposed Method

In this section, we describe S^3CMTF (Sparse, lock-free SGD based, and Scalable CMTF), our proposed method for fast, accurate, and scalable CMTF. CMTF

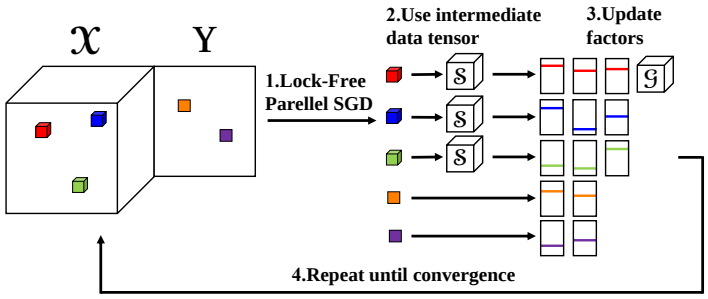


Fig. 2: The scheme for S^3CMTF.

methods for dense data are prone to getting high errors because of zero-filling for empty entries. On the other hand, CP-based methods show high prediction error because of simplicity of the model [13]. Our purpose is to devise an improved sparse CMTF model and propose a fast and scalable algorithm for the model.

We first propose a basic version of our method S^3CMTF-base; then, we propose a time-improved version S^3CMTF-opt. Figure 2 shows the overall scheme for S^3CMTF. S^3CMTF-base uses lock-free parallel SGD for the parallel update, and S^3CMTF-opt further improves the speed of S^3CMTF-base by exploiting intermediate data and reusing them.

3.1 Objective Function & Gradient

We discuss the improved formulation of the sparse CMTF problem defined in Definition 2. For simplicity, we consider the case that one matrix $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$ is coupled to the c-th mode of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. Equation (6) takes excessive time and memory since it includes materialization of dense tensor $\mathcal{G} \times \{\mathbf{U}\}$. Therefore, we formulate the new CMTF objective function f to exploit the sparsity of data. f is the weighted sum of two functions f_t and f_m where they are element-wise sums of squared reconstruction error and regularization terms of tensor $\mathcal{X}$ and matrix $\mathbf{Y}$, respectively.

$$f = \frac{1}{2} f_t + \frac{\lambda_m}{2} f_m \tag{7}$$

where λ_m is a balancing factor of the two functions.

$$f_t = \Big[\sum_{\forall \alpha \in \Omega_{\mathcal{X}}} \big(x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha)\big)^2 \Big] + \lambda_{reg} \Big(\|\mathcal{G}\|^2 + \sum_{n=1}^{N} \|\mathbf{U}^{(n)}\|^2 \Big)$$

where $\alpha = (i_1 \cdots i_N)$, $\Omega_{\mathcal{X}}$ is the observable index set of $\mathcal{X}$, and λ_{reg} denotes the regularization parameter for factors. We rewrite the equation so that it is amenable to SGD update.

$$f_t = \sum_{\forall \alpha \in \Omega_{\mathcal{X}}} \Big[\big(x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha)\big)^2 + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}|} \|\mathcal{G}\|^2 + \lambda_{reg} \sum_{n=1}^{N} \frac{\|\mathbf{u}_{i_n}^{(n)}\|^2}{|\Omega_{\mathcal{X}}^{n,i_n}|} \Big]$$

where $\alpha = (i_1 \cdots i_N)$. Note that $\Omega_{\mathcal{X}}^{n,i_n}$ is the subset of $\Omega_{\mathcal{X}}$ having i_n as the n-th index. Now we formulate f_m, the sum of squared errors of coupled matrix and

regularization term corresponding to the coupled matrix.

$$f_m = \sum_{\forall \beta=(j_1 j_2)\in \Omega_{\mathbf{Y}}} \Big[\big(y_\beta - \mathbf{u}^{(c)}_{j_1}\mathbf{v}^{\mathsf{T}}_{j_2}\big)^2 + \frac{\lambda_{reg}}{|\Omega^{2,j_2}_{\mathbf{Y}}|}\|\mathbf{v}_{j_2}\|^2 \Big]$$

We calculate the gradient of f (Equation (7)) with respect to factors for stochastic gradient descent update. Consider that we pick one index among tensor index $\alpha = (i_1 \cdots i_N) \in \Omega_{\mathcal{X}}$ and matrix index $\beta = (j_1 j_2) \in \Omega_{\mathbf{Y}}$. We calculate the corresponding partial derivatives of f with respect to the factors and the core tensor as follows.

$$\begin{aligned}
\left.\frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}}\right|_\alpha &= -\big(x_\alpha - (\mathcal{G}\times\{\mathbf{u}\}_\alpha)\big)\big[(\mathcal{G}\times_{-n}\{\mathbf{u}\}_\alpha)_{(n)}\big]^{\mathsf{T}} + \frac{\lambda_{reg}}{|\Omega^{n,i_n}_{\mathcal{X}}|}\mathbf{u}^{(n)}_{i_n} \\
\left.\frac{\partial f}{\partial \mathcal{G}}\right|_\alpha &= -\big(x_\alpha - (\mathcal{G}\times\{\mathbf{u}\}_\alpha)\big)\times\{\mathbf{u}\}^{\mathsf{T}}_\alpha + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}|}\mathcal{G} \\
\left.\frac{\partial f}{\partial \mathbf{u}^{(c)}_{j_1}}\right|_\beta &= -\lambda_m(y_\beta - \mathbf{u}^{(c)}_{j_1}\mathbf{v}^{\mathsf{T}}_{j_2})\mathbf{v}_{j_2} \\
\left.\frac{\partial f}{\partial \mathbf{v}_{j_2}}\right|_\beta &= -\lambda_m(y_\beta - \mathbf{u}^{(c)}_{j_1}\mathbf{v}^{\mathsf{T}}_{j_2})\mathbf{u}^{(c)}_{j_1} + \frac{\lambda_m\lambda_{reg}}{|\Omega^{2,j_2}_{\mathbf{Y}}|}\mathbf{v}_{j_2}
\end{aligned} \tag{8}$$

We omit the detailed derivation of Equations (8) for brevity. Note that our formulated coupled matrix-tensor factorization model is easily generalized to the case that multiple matrices are coupled to a tensor. We couple multiple matrices to a tensor for experiments in Sections 4 and 5.

3.2 Lock-Free Parallel Update

How can we parallelize the SGD updates in multiple cores? In general, SGD approach is hard to be parallelized because each parallel update may suffer from memory conflicts by attempting to write the same variables to memory concurrently [5]. One solution for this problem is memory locking and synchronization. However, there are lots of overhead associated with locking. Therefore, we use lock-free strategy to parallelize S^3CMTF. We develop parallel update scheme for S^3CMTF by adapting HOGWILD! update scheme [24].

Definition 3 (Induced Hypergraph) The objective function in Equation (7) induces a hypergraph $G = (V, E)$ whose nodes represent factor rows and core tensor. Each entry of $\mathcal{X}$ and $\mathbf{Y}$ induces a hyperedge $e \in E$ consisting of corresponding factor rows or core tensor. Figure 3a shows an example induced graph of S^3CMTF.

Lock-free parallel update guarantees near linear convergence property of a sparse SGD problem in which conflicts between different updates rarely occur [24]. However, in our formulation, every update of tensor entries includes the core tensor $\mathcal{G}$ as shown in Figure 3a. We allocate the update of core tensor $\mathcal{G}$ to one core to solve the problem. Then we obtain a new induced hypergraph in Figure 3b. The newly obtained hypergraph satisfies the sparsity condition for convergence. Lemma 1 proves the convergence property of parallel updates.

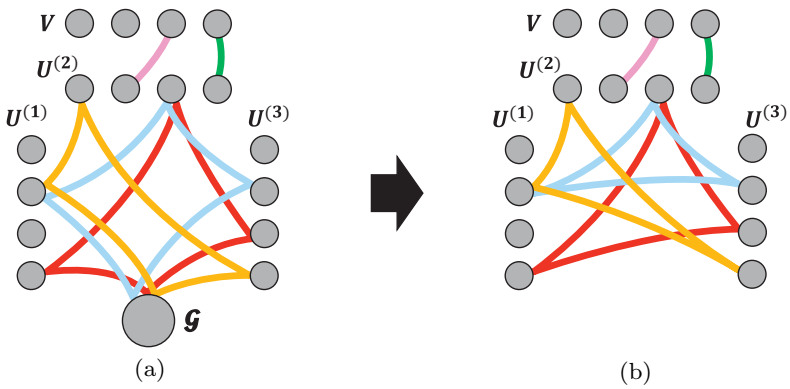


Fig. 3: Example graphs induced by S^3CMTF objective function (Equation (7)). A matrix $\mathbf{Y}$ is coupled to the second mode of $\mathcal{X}$ with a coupled factor matrix $\mathbf{V}$. Each node represents a factor row or the core tensor. Each hyperedge includes corresponding factors to an SGD update. (a) Induced hypergraph with core tensor. Every hyperedge corresponding to tensor entries includes $\mathcal{G}$. (b) Induced hypergraph without core tensor. The graph reveals sparsity as every node is shared by only few hyperedges.

Lemma 1 ***(Convergence)*** *If we assume that the elements of the tensor $\mathcal{X}$ and coupled matrix $\mathbf{Y}$ are sampled uniformly at random, lock-free parallel update of S^3CMTF converges to a local optimum.*

Proof For brevity, we assume that the dimension and rank of each mode are I and J, respectively. We use the notations used in Equation (2.6) of [24]. For a given hypergraph $G = (V, E)$, we define

$$\Omega := \max_{e \in E} |e|, \Delta := \frac{\max_{v \in V} |\{e \in E : v \in e\}|}{|E|}$$

$$\rho := \frac{\max_{e \in E} |\{\tilde{e} \in E : \tilde{e} \cap e \neq \emptyset\}|}{|E|}$$

First, consider the case when the tensor order is 2. Ω has the same value, and Δ has doubled value of the matrix factorization problem in [24]: $\Omega \approx 2J$, $\Delta \approx \frac{2\log(I)}{I}$. ρ naturally satisfies $\rho \approx \frac{3\log(I)}{I}$. Next, when the tensor order is N, Ω linearly scales up and $\Omega \approx NJ$, Δ and ρ stay same: $\Delta \approx \frac{2\log(I)}{I}$, $\rho \approx \frac{3\log(I)}{I}$. Parallel update converges as proved in Proposition 4.1 of [24].

3.3 S^3CMTF-base

We present a basic version of our method, S^3CMTF-base. S^3CMTF-base solves the sparse CMTF problem by parallel SGD techniques explained in Sections 3.1-3.2. Algorithm 1 shows the procedure of S^3CMTF-base. In the beginning, S^3CMTF-base initializes factor matrices and core tensor randomly (line 1 of Algorithm 1). The outer loop (lines 2-16) repeats until the factor variables converge.

Algorithm 1 S^3CMTF-base

Require: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, rank $(J_1, \cdots, J_N)$, number of parallel cores P, initial learning rate η_0, decay rate μ, coupled mode c, and coupled matrix $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$
Ensure: Core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, factor matrices $\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)}$, $\mathbf{V}$
1: Initialize $\mathcal{G}$, $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \cdots, N$, and $\mathbf{V}$ randomly
2: **repeat**
3: **for** $\forall \alpha = (i_1 \cdots i_N) \in \Omega_{\mathcal{X}}$, $\forall \beta = (j_1 j_2) \in \Omega_{\mathbf{Y}}$ in random order **do in parallel**
4: **if** α is picked **then**
5: $(\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}) \leftarrow$ *compute_gradient*$(\alpha, x_\alpha, \mathcal{G})$
6: $\mathbf{u}^{(n)}_{i_n} \leftarrow \mathbf{u}^{(n)}_{i_n} - \eta_t \frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}}$, (for $n = 1, \cdots, N$)
7: $\mathcal{G} \leftarrow \mathcal{G} - \eta_t P \frac{\partial f}{\partial \mathcal{G}}$ (executed by only one core)
8: **end if**
9: **if** β is picked **then**
10: $\tilde{y}_\beta \leftarrow \mathbf{u}^c_{j_1} \mathbf{v}^\top_{j_2}$, $\frac{\partial f}{\partial \mathbf{u}^{(c)}_{j_1}} \leftarrow -\lambda_m (y_\beta - \tilde{y}_\beta) \mathbf{v}_{j_2}$
11: $\frac{\partial f}{\partial \mathbf{v}_{j_2}} \leftarrow -\lambda_m (y_\beta - \tilde{y}_\beta) \mathbf{u}^{(c)}_{j_1} + \frac{\lambda_m \lambda_{reg}}{|\Omega_{\mathbf{Y}_{2,j_2}}|} \mathbf{v}_{j_2}$
12: $\mathbf{u}^{(c)}_{j_1} \leftarrow \mathbf{u}^{(c)}_{j_1} - \eta_t \frac{\partial f}{\partial \mathbf{u}^{(c)}_{j_1}}$, $\mathbf{v}_{j_2} \leftarrow \mathbf{v}_{j_2} - \eta_t \frac{\partial f}{\partial \mathbf{v}_{j_2}}$
13: **end if**
14: **end for**
15: $\eta_t = \eta_0 (1 + \mu t)^{-1}$
16: **until** convergence conditions are satisfied
17: **for** $n = 1, \ldots, N$ **do**
18: $\mathbf{Q}^{(n)}, \mathbf{R}^{(n)} \leftarrow$ QR decomposition of $\mathbf{U}^{(n)}$
19: $\mathbf{U}^{(n)} \leftarrow \mathbf{Q}^{(n)}$, $\mathcal{G} \leftarrow \mathcal{G} \times_n \mathbf{R}^{(n)}$
20: **end for**
21: $\mathbf{V} \leftarrow \mathbf{V}\mathbf{R}^{(c)\top}$
22: **return** $\mathcal{G}$, $\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)}, \mathbf{V}$

Algorithm 2 *compute_gradient*$(\alpha, x_\alpha, \mathcal{G})$

Require: Tensor entry x_α, $\alpha = (i_1 \cdots i_N) \in \Omega_{\mathcal{X}}$, core tensor $\mathcal{G}$
Ensure: Gradients $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$
1: $\tilde{x}_\alpha \leftarrow \mathcal{G} \times \{\mathbf{u}\}_\alpha$
2: **for** $n = 1, \cdots, N$ **do**
3: $\frac{\partial f}{\partial \mathbf{u}^{(n)}_i} \leftarrow -\big(x_\alpha - \tilde{x}_\alpha\big) \big[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}\big]^\top + \frac{\lambda_{reg}}{|\Omega^{n,i_n}_{\mathcal{X}}|} \mathbf{u}^{(n)}_{i_n}$
4: **end for**
5: $\frac{\partial f}{\partial \mathcal{G}} \leftarrow -\big(x_\alpha - \tilde{x}_\alpha\big) \times \{\mathbf{u}\}^\top_\alpha + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}|} \mathcal{G}$
6: **return** $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

The inner loop (lines 3-14) is performed by several cores in parallel except for line 7. In each inner loop, S^3CMTF-base selects an index which belongs to $\Omega_{\mathcal{X}}$ or $\Omega_{\mathbf{Y}}$ in random order (line 3). If a tensor index α is picked, then the algorithm calculates the partial gradients of corresponding factor rows using *compute_gradient* (Algorithm 2) in line 5, and updates factor row vectors (line 6). Core tensor $\mathcal{G}$ is updated by only one core (line 7); the number P of cores is multiplied to the gradient to compensate for the one-core update so that SGD uses the same learning rate for all the parameters. If a coupled matrix index β is picked, then the gradient update is performed on corresponding factor row vectors (lines 9-13). At the end

Algorithm 3 *compute_gradient_opt*(α,x_α,$\mathcal{G}$)

Require: Tensor entry x_α, $\alpha = (i_1 \cdots i_N) \in \Omega_{\mathcal{X}}$, core tensor $\mathcal{G}$
Ensure: Gradients $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

$\tilde{x}_\alpha \leftarrow 0$
for $\forall (j_1 j_2 \cdots j_N) \in \Omega_{\mathcal{G}}$ **do**
  $s_{j_1 j_2 \cdots j_N} \leftarrow g_{j_1 j_2 \cdots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N}$
  $\tilde{x}_\alpha \leftarrow \tilde{x}_\alpha + s_{j_1 j_2 \cdots j_N}$
end for
for $n = 1, \ldots, N$ **do**
  $\frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot Collapse(\mathcal{S}, n) \oslash \mathbf{u}^{(\mathbf{n})}_{\mathbf{i_n}} + \frac{\lambda_{reg}}{|\Omega^{n,i_n}_{\mathcal{X}}|} \mathbf{u}^{(\mathbf{n})}_{\mathbf{i_n}}$
end for
$\frac{\partial f}{\partial \mathcal{G}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot \mathcal{S} \oslash \mathcal{G} + \lambda_{reg} \mathcal{G}$
return $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \ldots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

of the outer loop, the learning rate η_t is monotonically decreased [4]. (line 15). QR decomposition is applied on factors to satisfy orthogonality constraint of factor matrices (lines 17-20). QR decomposition of $\mathbf{U}^{(n)}$ generates $\mathbf{Q}^{(n)}$, an orthogonal matrix of the same size as $\mathbf{U}^{(n)}$, and a square matrix $\mathbf{R}^{(n)} \in \mathbb{R}^{J_n \times J_n}$. Substituting $\mathbf{U}^{(n)}$ by $\mathbf{Q}^{(n)}$ (line 19) and $\mathcal{G}$ by $\mathcal{G} \times_1 \mathbf{R}^{(1)} \cdots \times_N \mathbf{R}^{(N)}$ (after N-th execution of line 19) result in an equivalent factorization [14]. In the same manner, we substitute $\mathbf{V}$ by $\mathbf{V}\mathbf{R}^{(c)\mathsf{T}}$ (line 21) because $\tilde{\mathbf{Y}} = \mathbf{U}^{(c)}\mathbf{V}^{\mathsf{T}} = \mathbf{Q}^{(c)}\mathbf{R}^{(c)}\mathbf{V}^{\mathsf{T}} = \mathbf{Q}^{(c)}(\mathbf{V}\mathbf{R}^{(c)\mathsf{T}})^{\mathsf{T}}$.

3.4 S^3CMTF-opt

There are many redundant calculations in S^3CMTF-base. For example, $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ is calculated for every execution of *compute_gradient* (Algorithm 2) in line 5 of Algorithm 1. In S^3CMTF-opt, we save the time by storing the intermediate calculation for $\tilde{x}_\alpha$ and reusing them.

Definition 4 (Intermediate Data) When updating the factor rows for a tensor entry $x_{\alpha=(i_1 \cdots i_N)}$, we define $(j_1 j_2 \cdots j_N)$-th element of intermediate data $\mathcal{S}$:

$$s_{j_1 j_2 \cdots j_N} \leftarrow g_{j_1 j_2 \cdots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N}$$

There is no extra time required for calculating $\mathcal{S}$ because $\mathcal{S}$ is generated while calculating $\tilde{x}_\alpha$. Lemma 2 shows that $\tilde{x}_\alpha$ is calculated by summing all entries of $\mathcal{S}$.

Lemma 2 *For a given tensor index α, estimated tensor entry $\tilde{x}_\alpha = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} s_{j_1 j_2 \cdots j_N}$.*

Proof The proof is straightforward by Equation (4).

We use $\mathcal{S}$ with following *Collapse* operation to calculate gradients efficiently.

Definition 5 (Collapse) The *Collapse* operation of the intermediate tensor $\mathcal{S}$ on the n-th mode outputs a row vector defined as

$$Collapse(\mathcal{S}, n) = \left[\sum_{\forall \delta \in \Omega^{n,1}_{\mathcal{S}}} s_\delta, \sum_{\forall \delta \in \Omega^{n,2}_{\mathcal{S}}} s_\delta, \cdots, \sum_{\forall \delta \in \Omega^{n,J_n}_{\mathcal{S}}} s_\delta \right]$$

Collapse operation aggregates the elements of intermediate tensor $\mathcal{S}$ with respect to a fixed mode. We re-express the calculation of gradients for tensor factors in Equations (8) in an efficient manner.

Lemma 3 ***(Efficient Gradient Calculation)*** *Followings are equivalent calculations of tensor factors gradients as Equations* (8)*.*

$$\tilde{x}_\alpha \leftarrow \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} s_{j_1 j_2 \cdots j_N} \tag{9}$$

$$\frac{\partial f}{\partial \mathbf{u}_{i_n}^{(n)}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot Collapse(\mathcal{S}, n) \oslash \mathbf{u}_{i_n}^{(n)} + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}^{n,i_n}|} \mathbf{u}_{i_n}^{(n)} \tag{10}$$

$$\frac{\partial f}{\partial \mathcal{G}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot \mathcal{S} \oslash \mathcal{G} + \lambda_{reg} \mathcal{G} \tag{11}$$

where $\alpha = (i_1 i_2 \cdots i_N)$ *and* $\oslash$ *is element-wise division.*

Proof In Lemma 2, Equation (9) is proved. To prove the equivalence of Equation (10) and the first equation of Equations (8), it suffices to show $[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]^\mathsf{T} = Collapse(\mathcal{S}, n) \oslash \mathbf{u}_{i_n}^{(n)}$ where $\alpha = (i_1 \cdots i_N) \in \Omega_{\mathcal{X}}$ and $\delta = (j_1 \cdots j_N) \in \Omega_{\mathcal{G}}^{n,k}$. We use Equation (3) for the proof.

$$[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]_k^\mathsf{T} = \sum_{\forall \delta \in \Omega_{\mathcal{G}}^{n,k}} g_\delta u_{i_1 j_1}^{(1)} \cdots u_{i_{n-1} j_{n-1}}^{(n-1)} u_{i_{n+1} j_{n+1}}^{(n+1)} \cdots u_{i_N j_N}^{(N)}$$

$$= \sum_{\forall \delta \in \Omega_{\mathcal{G}}^{n,k}} g_\delta u_{i_1 j_1}^{(1)} \cdots u_{i_{n-1} j_{n-1}}^{(n-1)} u_{i_n k}^{(n)} u_{i_{n+1} j_{n+1}}^{(n+1)} \cdots u_{i_N j_N}^{(N)} / u_{i_n k}^{(n)}$$

$$= \sum_{\forall \delta \in \Omega_{\mathcal{S}}^{n,k}} s_\delta / u_{i_n k}^{(n)} = \frac{[Collapse(\mathcal{S}, n)]_k}{u_{i_n k}^{(n)}} = [Collapse(\mathcal{S}, n) \oslash \mathbf{u}_{i_n}^{(n)}]_k$$

Next, to show the equivalence of Equation (11) and the second equation of Equations (8), it suffices to show $1 \times \{\mathbf{u}\}_\alpha^\mathsf{T} = \mathcal{S} \oslash \mathcal{G}$.

$$[1 \times \{\mathbf{u}\}_\alpha^\mathsf{T}]_{\gamma = (l_1 l_2 \cdots l_N)} = u_{i_1 l_1}^{(1)} u_{i_2 l_2}^{(2)} \cdots u_{i_N l_N}^{(N)}$$

$$= g_\gamma u_{i_1 l_1}^{(1)} \cdots u_{i_N l_N}^{(N)} / g_\gamma = s_\gamma / g_\gamma = [\mathcal{S} \oslash \mathcal{G}]_\gamma$$

S^3CMTF-opt replaces *compute_gradient* (Algorithm 2) of S^3CMTF-base with *compute_gradient_opt* (Algorithm 3), a time-improved alternative using Lemma 3. We prove that the new calculation scheme is faster than the previous one.

Lemma 4 *compute_gradient_opt is faster than compute_gradient. The theoretical time complexity of compute_gradient is* $\mathcal{O}(N^2 J^N)$ *and the time complexity of compute_gradient_opt is* $\mathcal{O}(N J^N)$ *where* $J_1 = J_2 = \cdots = J_N = J$*.*

Proof We assume that $I_1 = I_2 = \cdots = I_N = I$ for brevity. First, we calculate the time complexity of *compute_gradient* (Algorithm 2). Given a tensor index α, computing $\tilde{x}_\alpha$ (line 1 of Algorithm 2) takes $\mathcal{O}(NJ^N)$. Computing $(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)$ (line 3) takes $\mathcal{O}(NJ^N)$. Thus, aggregate time for calculating the row gradient for all modes (lines 2-4) takes $\mathcal{O}(N^2 J^N)$. Calculating $(x_\alpha - \tilde{x}_\alpha) \times \{\mathbf{u}\}_\alpha^\mathsf{T}$ (line 5) takes $\mathcal{O}(NJ^N)$. In sum, *compute_gradient* takes $\mathcal{O}(N^2 J^N)$ time. Next, we calculate the time complexity of *compute_gradient_opt* (Algorithm 3). Computing an entry of intermediate data $\mathcal{S}$ (line 3 of Algorithm 3) takes $\mathcal{O}(N)$. Aggregate time for getting $\mathcal{S}$ (lines 2-5) is $\mathcal{O}(NJ^N)$ because $|\Omega_{\mathcal{G}}| = \mathcal{O}(J^N)$. Calculating row gradient for all modes (lines 6-8) takes $\mathcal{O}(NJ^N)$ because *Collapse* operation takes $\mathcal{O}(J^N)$. Calculating gradient for core tensor (line 9) takes $\mathcal{O}(J^N)$. In sum, *compute_gradient_opt* takes $\mathcal{O}(NJ^N)$ time.

Table 3: Comparison of time complexity (per iteration) and memory usage of our proposed S^3CMTF and other CMTF algorithms. S^3CMTF-opt shows the lowest time complexity and S^3CMTF-base shows the lowest memory usage. For simplicity, we assume that all modes are of size I, of rank J, and an $I \times K$ matrix is coupled to one mode. P is the number of parallel cores. (* indicates the lowest time or memory.)

	Time complexity (per iter.)	**Memory usage**
S^{3}CMTF-base	$\mathcal{O}(\lvert\Omega_{\mathcal{X}}\rvert N^2 J^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$	$\mathcal{O}(PJ)$*
S^{3}CMTF-opt	$\mathcal{O}(\lvert\Omega_{\mathcal{X}}\rvert N J^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$*	$\mathcal{O}(PJ^N)$
CMTF-Tucker-ALS	$\mathcal{O}(NI^{N-1}J^2 + NI^2J^{N-1} + I^2K)$	$\mathcal{O}(IJ^{N-1})$
CMTF-OPT	$\mathcal{O}(\lvert\Omega_{\mathcal{X}}\rvert NJ + NI^{N-1}J + IJK)$	$\mathcal{O}(I^{N-1}J + JK)$

3.5 Analysis

We analyze the proposed method in terms of time complexity per iteration. For simplicity, we assume that $I_1 = I_2 = \cdots = I_N = I$, and $J_1 = J_2 = \cdots = J_N = J$. Table 3 summarizes the time complexity (per iteration) and memory usage of S^3CMTF and other methods. Note that the memory usage refers to the auxiliary space for temporary variables used by a method.

Lemma 5 *The time complexity (per iteration) of S^3CMTF-base is $\mathcal{O}(\lvert\Omega\rvert N^2 J^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$ and the time complexity (per iteration) of S^3CMTF-opt is $\mathcal{O}(\lvert\Omega\rvert N J^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$ where P denotes the number of parallel cores.*

Proof First, we check the time complexity of S^3CMTF-base (Algorithm 1). When a tensor index α is picked in the inner loop (line 4 of Algorithm 1), calculating gradients with respect to tensor factors (line 5) takes $\mathcal{O}(N^2J^N)$ as shown in Lemma 4. Updating factor rows (line 6) takes $\mathcal{O}(NJ)$, and updating core tensor (line 7) takes $\mathcal{O}(J^N)$. If a coupled matrix index β is picked (line 9), calculating $\tilde{y}_\beta$ (line 10) takes $\mathcal{O}(J)$. Calculating and updating the factor rows corresponding to coupled matrix entry (lines 10-12) take $\mathcal{O}(J)$. All calculations except updating core tensor (line 7) are conducted in parallel. Finally, for all $\alpha \in \Omega_{\mathcal{X}}$ and $\beta \in \Omega_{\mathbf{Y}}$, S^3CMTF-base takes $\mathcal{O}(\lvert\Omega_{\mathcal{X}}\rvert N^2 J^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$ for one iteration. S^3CMTF-opt uses *compute_gradient_opt* instead of *compute_gradient* in line 5 of Algorithm 1, whose time complexity is shown in Lemma 4. Overall running time per iteration for S^3CMTF-opt is $\mathcal{O}(\lvert\Omega_{\mathcal{X}}\rvert NJ^N/P + \lvert\Omega_{\mathbf{Y}}\rvert J/P)$.

4 Experiments

In this and the next sections, we experimentally evaluate S^3CMTF. Especially, we answer the following questions.
Q1 : **Performance (Section 4.2)** How accurate and fast is S^3CMTF compared to competitors?
Q2 : **Scalability (Section 4.3)** How do S^3CMTF and other methods scale in terms of dimensionality, the number of observed entries, and the number of cores?
Q3 : **Discovery (Section 5)** What are the discoveries of applying S^3CMTF on real-world data?

Table 4: Summary of the data used for experiments. K: thousand, and M: million. Data of density 1 are fully observed.

Name	Data	Dimensionality	# entries	Density
MovieLens	User-Movie-Time	71K-11K-157	10M	$\sim 10^{-4}$
	Movie-Genre	20	214K	1
Netflix	User-Movie-Time	480K-18K-74	100M	$\sim 10^{-4}$
	Movie-Yearmonth	110	2M	1
Yelp	User-Business-Time	1M-144K-149	4M	$\sim 10^{-7}$
	User-User	1M	7M	$\sim 10^{-4}$
	Business-Category	1K	172M	1
	Business-City	1K	126M	1
Synthetic	3-mode tensor	1K$\sim$100M	1K$\sim$100M	$10^{-20\sim-3}$
	Matrix	1K$\sim$100M	1K$\sim$100M	$10^{-11\sim-4}$

4.1 Experimental Settings

Data. Table 4 shows the data we used in our experiments. We use three real-world datasets (MovieLens[1], Netflix[2], and Yelp[3]) and generate synthetic data to evaluate S^3CMTF. Each entry of the real-world datasets represents a rating, which consists of (user, 'item', time; rating) where 'item' indicates 'movie' for MovieLens and Netflix, and 'business' for Yelp. We use (movie, genre) and (movie, year) as coupled matrices for MovieLens and Netflix, respectively. We use (user, user) friendship matrix, (business, category) and (business, city) matrices for Yelp. We generate 3-mode synthetic random tensors with dimensionality I and corresponding coupled matrices. We vary I in the range of 1K$\sim$100M and the number of tensor entries in the range of 1K$\sim$100M. We set the number of entries as $|\Omega_{\mathbf{Y}}| = \frac{1}{10}|\Omega_{\mathcal{X}}|$ for synthetic coupled matrices.

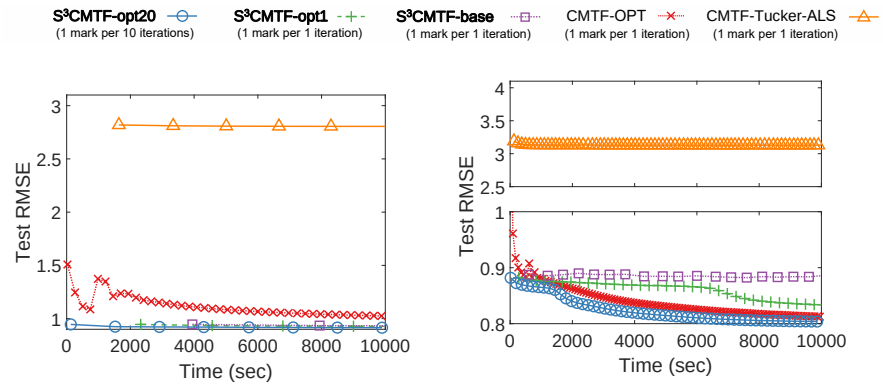
Measure. We use test RMSE as the measure for tensor reconstruction error.

$$\text{test RMSE} = \sqrt{\frac{1}{|\Omega_{test}|} \sum_{\forall \alpha \in \Omega_{test}} (x_\alpha - \tilde{x}_\alpha)^2}$$

where Ω_{test} is the index set of the test data tensor, x_α stands for each test tensor entry, and $\tilde{x}_\alpha$ is the corresponding reconstructed value.

Methods. We compare S^3CMTF-base and S^3CMTF-opt with other single machine CMTF methods: CMTF-Tucker-ALS and CMTF-OPT (described in Section 2.3). To examine multi-core performance, we run two versions of S^3CMTF-opt: S^3CMTF-opt1 (1 core), and S^3CMTF-opt20 (20 cores). We exclude distributed CMTF methods [9, 11, 3] since they are designed for Hadoop with multiple machines, and thus take too much time for single machine environment. For example, [20] reported that HaTen2 [11] takes 10,700s to decompose 4-way tensor with $I = 10K$ and $|\Omega_{\mathcal{X}}| = 100K$, which is almost 7,000$\times$ slower than our single machine implementation of S^3CMTF-opt. For CMTF-Tucker-ALS, we use a C++ implementation based on Tucker-MET [16], and for CMTF-OPT, we use a C++ implementation of CMTF-OPT [1]. We implement S^3CMTF with C++, and OpenMP library for multi-core parallelization. The codes and datasets used in this paper are available at https://datalab.snu.ac.kr/S3CMTF.

[1] http://grouplens.org/datasets/movielens/10m
[2] http://www.netflixprize.com
[3] http://www.yelp.com/dataset_challenge



(a) **Netflix** (b) **MovieLens**

Fig. 4: Test RMSE of S^3CMTF and other CMTF methods over iterations. S^3CMTF-opt20 shows the best convergence rate and accuracy.

We conduct all experiments on a machine equipped with Intel Xeon E5-2630 v4 2.2GHz CPU and 256GB RAM. We mark out-of-memory (O.O.M.) error when the memory usage exceeds the limit.

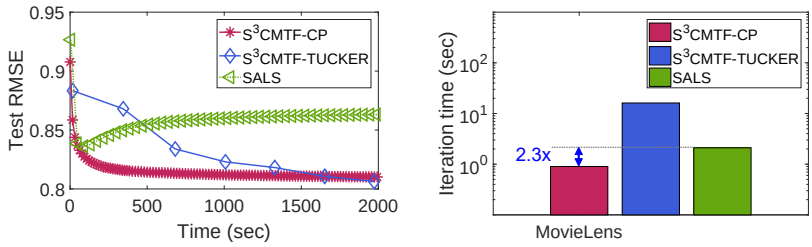
Parameters. We set pre-defined parameters: tensor rank J, regularization factor λ_{reg}, λ_m, the initial learning rate η_0, and decay rate μ. We set λ_{reg} to 0.1, $\lambda_m = 10$, and $\mu = 0.1$ for all datasets. For rank and initial learning rate, MovieLens: $J = 12, \eta = 0.001$, Netflix: $J = 11, \eta = 0.001$, and Yelp: $J = 10, \eta = 0.0005$.

4.2 Performance of S^3CMTF

We observe the performance of S^3CMTF to answer Q1. As seen in Figures 1b and 4, S^3CMTF converges faster to the optimum with the lowest test error than existing methods with the following details.

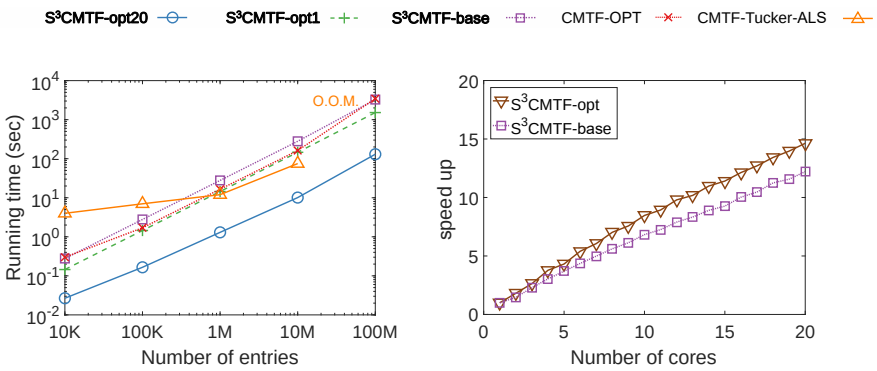
Accuracy. We divide each data tensor into 80%/20% for train/test sets. The lower error for a same elapsed time implies the better accuracy and faster convergence. Figures 1b and 4 show the changes of test RMSE of each method on three datasets over elapsed time which are the answers for Q1. S^3CMTF achieves the lowest error compared to others for the same elapsed time. For Yelp, CMTF-Tucker-ALS shows O.O.M. error. S^3CMTF-opt20 achieves the lowest error 1.253, 0.9147, and 0.8037 while the best competing method CMFT-OPT gives the error 1.370, 1.018, and 0.8125 for Yelp, Netflix, and MovieLens datasets, respectively. Note that another competing method CMFT-Tucker-ALS gives an out of memory error or highest error rate.

Running time. We compare our method with the multi-core version of SALS-single [27], a parallel CP decomposition algorithm, to demonstrate the high performance of S^3CMTF compared to up-to-date decomposition algorithms. We used non-coupled CP version of our method, S^3CMTF-CP, by setting $\mathcal{G}$ to be hyper-diagonal and not coupling any matrices. Figure 5 shows that S^3CMTF is better than SALS-single in terms of both error and time for MovieLens dataset. S^3CMTF-TUCKER denotes the non-coupled version of S^3CMTF-opt.



(a) **Convergence comparison** (b) **Time comparison**

Fig. 5: Comparison with SALS-single. We compare two non-coupled version of S^3CMTF, S^3CMTF-CP and S^3CMTF-TUCKER with the parallel CP decomposition method, SALS-single. For (a), we set 1 mark per 20 iterations for clarity. (a) S^3CMTF converges faster to a lower error than SALS does. (b) S^3CMTF-CP is 2.3× faster than SALS-single.



(a) **Running time vs. Number of entries** (b) **Parallel scalability**

Fig. 6: Comparison of scalability. (a) S^3CMTF shows linear scalability as the number of entries increases. (b) S^3CMTF-base and S^3CMTF-opt show linear *speed up* as the number of cores grows. O.O.M.: out of memory error.

4.3 Scalability Analysis

We present scalability of our proposed S^3CMTF and comeptitors to answer Q2, in terms of two aspects: data scalability and parallel scalability. We use synthetic data of varying size for evaluation. As a result, we show the running time (for one iteration) of S^3CMTF follows our theoretical analysis in Section 3.5.

Data Scalability. The time complexity of CMTF-Tucker-ALS and CMTF-OPT have $\mathcal{O}(NI^{N-1}J^2)$ and $\mathcal{O}(NI^{N-1}J)$ as their dominant terms, respectively. In contrast, S^3CMTF exploits the sparsity of input data, and has the time complexity linear to the number of entries ($|\Omega_{\mathcal{X}}|$, $|\Omega_{\mathbf{Y}}|$) and independent to the dimensionality (I) as shown in Lemma 5. Figures 1a and 6 show that the running time (for one iteration) of S^3CMTF follows our theoretical analysis in Section 3.5. First,

we fix $|\Omega_{\mathcal{X}}|$ to 1M and $|\Omega_{\mathbf{Y}}|$ to 100K, and vary dimensionality I from 1K to 100M. Figure 1a shows the running time (for one iteration) of all methods. Note that all of our proposed methods achieve constant running time as dimensionality increases because they exploit the sparsity of data by updating factors related to only observed data entries. However, CMTF-Tucker-ALS and CMTF-OPT show exponentially increasing running time, and CMTF-OPT shows O.O.M. when $I = 10M$. Next, we investigate the data scalability over the number of entries as shown in Figure 6a. We fix I to 10K and raise $|\Omega_{\mathcal{X}}|$ from 10K to 100M. CMTF-Tucker-ALS shows O.O.M. when $|\Omega_{\mathcal{X}}| = 100M$, and CMTF-OPT shows near-linear scalability. Focusing on the results of S^3CMTF, all three versions of our approach show linear relation between running time and $|\Omega_{\mathcal{X}}|$.

Parallel Scalability. We conduct experiments to examine parallel scalability of S^3CMTF on shared memory systems. For measurement, we define *speed up* as *(iteration time on 1 core)/(iteration time)*. Figure 6b shows the linear *speed up* of S^3CMTF-base and S^3CMTF-opt. S^3CMTF-opt earns higher *speed up* than S^3CMTF-base because it reduces reading accesses for core tensor by utilizing intermediate data.

5 Discovery

In this section, we use S^3CMTF for mining real-world data, Yelp, to answer the question Q3 in the beginning of Section 4. First, we demonstrate that S^3CMTF has better discernment for business entities compared to the naive decomposition method by jointly capturing spatial and categorical prior knowledge. Second, we show how S^3CMTF is possibly applied to the real recommender systems. It is an open challenge to jointly capture the spatio-temporal context along with user preference data [8]. We exemplify a personal recommendation for a specific user. For discovery, we use the total Yelp data tensor along with coupled matrices as explained in Table 4. For better interpretability, we found non-negative factorization by applying projected gradient method [18]. Orthogonality condition is not imposed to keep non-negativity, and each column of factors is normalized.

Discovery. First, we compare discernment by S^3CMTF and the Tucker decomposition. We use the business factor $\mathbf{U}^{(2)}$. Figure 7a shows gap statistic values of clustering business entities with k-means clustering algorithm. S^3CMTF shows higher gap statistic value compared to the Tucker decomposition which means S^3CMTF outperforms the naive Tucker decomposition for entity clustering.

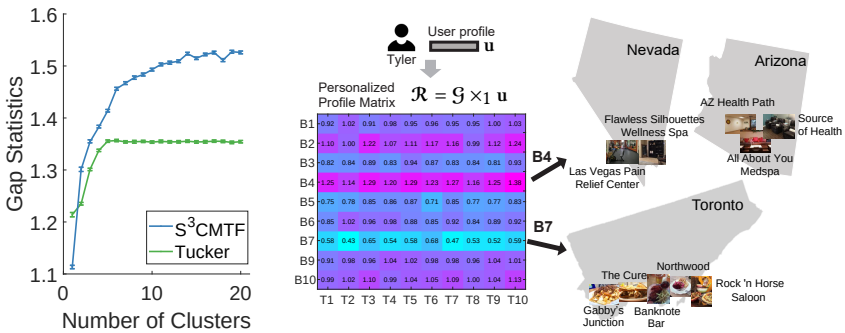
As the difference between S^3CMTF and the Tucker decomposition is the existence of coupled matrices, the high performance of S^3CMTF is attributed to the unified factorization using spatial and categorical data as prior knowledge. Table 5 shows the found clusters of business entities. Note that each cluster represents a certain combination of spatial and categorical characteristics of business entities.

User-specific recommendation. Commercial recommendation is one of the most important applications of factorization models [17, 12]. Here we illustrate how factor matrices are used for personalized recommendations with a real example. Figure 7b shows the process for recommendation. Below, we illustrate the process in detail.

- An example user Tyler has a factor vector $\mathbf{u}$, namely user profile, which has been calculated by previous review histories.

Table 5: Clustering results on business factor $\mathbf{U}^{(2)}$ found by S^3CMTF. We found dominant spatial and categorical characteristics from each cluster. Businesses in a same cluster tend to be in adjacent cities and are included in similar categories.

Cluster	Location / Category	Top-10 Businesses
C1	Las Vegas, US/ Travel & Entertainment	Nocturnal Tours, Eureka Casino, Happi Inn, Planet Hollywood Poker Room, Circus Midway Arcade, etc.
C2	Arizona, US/ Real estate & Home services	ENMAR Hardwood Flooring, Sprinkler Dude LLC, Eklund Refrigeration, NR Quality Handyman, The Daniel Montez Real Estate Group, etc.
C11	Ontario, Canada/ Restaurants & Deserts	Jyuban Ramen House, Tim Hortons, Captain John Donlands Fish and Chips, Cora's Breakfast & Lunch, Pho Pad Thai, etc.
C17	Ohio, US/ Food & Drinks	ALDI, Pulp Juice and Smoothie Bar, One Barrel Brewing, Wok N Roll Food Truck, Gas Pump Coffee Company, etc.



(a) **Gap Statistics** (b) **Personal recommendation**

Fig. 7: (a) Gap statistics on $\mathbf{U}^{(2)}$ of S^3CMTF and the Tucker decomposition for Yelp dataset. S^3CMTF outperforms the naive Tucker decomposition for its clustering ability. (b) Visualization of the personal recommendation scenario.

- We then calculate the personalized profile matrix $\mathcal{R} = \mathcal{G} \times_1 \mathbf{u} (\in \mathbb{R}^{J_2 \times J_3})$. $\mathcal{R}$ measures the amount of interaction of user profile with business and time factors.
- Norm values of rows in $\mathcal{R}$ indicate the influence of latent business concepts on Tyler. Dominant and weak concepts are found based on the calculated norm values. In the example, B4 is the dominant, and B7 is the weak latent concept.
- We inspect the corresponding columns of business factor matrix $\mathbf{U}^{(2)}$ and find relevant business entities with high values for the found concepts (B4 and B7).

We found both strong and weak entities by the above process. The strong and weak entities provide recommendation information by themselves in the sense that

the probability of the user to like strong and weak entities are high and low, respectively, and they also give extended user preference information. For example, strong entities for Tyler are related to 'spa & health' and located in neighborhood cities of Arizona, US. Weak entities are related to 'grill & restaurants' and located in Toronto, Canada. The captured user preference information potentially makes commercial recommender systems more powerful with additional user-specific information such as address, current location, etc.

6 Conclusion

We propose S^3CMTF, a fast, accurate, and scalable CMTF method. S^3CMTF provides 989× faster running time and the best accuracy by sparse Tucker factorization with carefully derived update rules, lock-free parallel SGD, and reusing intermediate computation results. S^3CMTF shows linear scalability for the number of data entries and parallel cores. Moreover, we show the usefulness of S^3CMTF for cluster analysis and recommendation by applying S^3CMTF to real-world Yelp data. Future works include extending the method to a distributed setting.

References

1. Acar, E., Kolda, T.G., Dunlavy, D.M.: All-at-once optimization for coupled matrix and tensor factorizations. arXiv preprint arXiv:1105.3422 (2011)
2. Acar, E., Rasmussen, M.A., Savorani, F., Næs, T., Bro, R.: Understanding data fusion within the framework of coupled matrix and tensor factorizations. Chemometrics and Intelligent Laboratory Systems **129**, 53–63 (2013)
3. Beutel, A., Talukdar, P.P., Kumar, A., Faloutsos, C., Papalexakis, E.E., Xing, E.P.: Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In: Proceedings of the 2014 SIAM International Conference on Data Mining, pp. 109–117. SIAM (2014)
4. Bottou, L.: Stochastic gradient descent tricks. In: Neural networks: Tricks of the trade, pp. 421–436. Springer (2012)
5. Bradley, J.K., Kyrola, A., Bickson, D., Guestrin, C.: Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379 (2011)
6. De Lathauwer, L., De Moor, B., Vandewalle, J.: On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. SIAM journal on Matrix Analysis and Applications **21**(4), 1324–1342 (2000)
7. Ding, C., Li, T., Peng, W.: On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. Computational Statistics & Data Analysis **52**(8), 3913–3927 (2008)
8. Gao, H., Tang, J., Hu, X., Liu, H.: Exploring temporal effects for location recommendation on location-based social networks. In: Proceedings of the 7th ACM conference on Recommender systems, pp. 93–100. ACM (2013)
9. Jeon, B., Jeon, I., Sael, L., Kang, U.: Scout: Scalable coupled matrix-tensor factorization-algorithm and discoveries. In: Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, pp. 811–822. IEEE (2016)
10. Jeon, I., Papalexakis, E.E., Faloutsos, C., Sael, L., Kang, U.: Mining billion-scale tensors: algorithms and discoveries. The VLDB Journal **25**(4), 519–544 (2016)
11. Jeon, I., Papalexakis, E.E., Kang, U., Faloutsos, C.: Haten2: Billion-scale tensor decompositions. In: Data Engineering (ICDE), 2015 IEEE 31st International Conference on, pp. 1047–1058. IEEE (2015)
12. Karatzoglou, A., Amatriain, X., Baltrunas, L., Oliver, N.: Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In: Proceedings of the fourth ACM conference on Recommender systems, pp. 79–86. ACM (2010)
13. Kiers, H.A., Ten Berge, J.M., Rocci, R.: Uniqueness of three-mode factor models with sparse cores: The 3× 3× 3 case. Psychometrika **62**(3), 349–374 (1997)

14. Kolda, T.G.: Multilinear operators for higher-order decompositions. Tech. rep., Sandia National Laboratories (2006)
15. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM review **51**(3), 455–500 (2009)
16. Kolda, T.G., Sun, J.: Scalable tensor decompositions for multi-aspect data mining. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pp. 363–372. IEEE (2008)
17. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8) (2009)
18. Lin, C.J.: Projected gradient methods for nonnegative matrix factorization. Neural computation **19**(10), 2756–2779 (2007)
19. Narita, A., Hayashi, K., Tomioka, R., Kashima, H.: Tensor factorization using auxiliary information. Data Mining and Knowledge Discovery **25**(2), 298–324 (2012)
20. Oh, J., Shin, K., Papalexakis, E.E., Faloutsos, C., Yu, H.: S-hot: Scalable high-order tucker decomposition. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp. 761–770. ACM (2017)
21. Ozcaglar, C.: Algorithmic data fusion methods for tuberculosis. Ph.D. thesis, Rensselaer Polytechnic Institute (2012)
22. Papalexakis, E.E., Faloutsos, C., Mitchell, T.M., Talukdar, P.P., Sidiropoulos, N.D., Murphy, B.: Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In: Proceedings of the 2014 SIAM International Conference on Data Mining, pp. 118–126. SIAM (2014)
23. Peng, W., Li, T.: On the equivalence between nonnegative tensor factorization and tensorial probabilistic latent semantic analysis. Applied Intelligence **35**(2), 285–295 (2011)
24. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems, pp. 693–701 (2011)
25. Rendle, S., Schmidt-Thieme, L.: Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the third ACM international conference on Web search and data mining, pp. 81–90. ACM (2010)
26. Sael, L., Jeon, I., Kang, U.: Scalable tensor mining. Big Data Research **2**(2), 82–86 (2015)
27. Shin, K., Sael, L., Kang, U.: Fully scalable methods for distributed tensor factorization. IEEE Transactions on Knowledge and Data Engineering **29**(1), 100–113 (2017)
28. Xu, W., Liu, X., Gong, Y.: Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, pp. 267–273. ACM (2003)