D-Tucker: Fast and Memory-Efficient Tucker Decomposition for Dense Tensors

Jun-Gi Jang
Computer Science and Engineering
Seoul National University
elnino4@snu.ac.kr

U Kang
Computer Science and Engineering
Seoul National University
ukang@snu.ac.kr

***Abstract*—Given a dense tensor, how can we find latent patterns and relations efficiently? Existing Tucker decomposition methods based on Alternating Least Square (ALS) have limitations in terms of time and space since they directly handle large dense tensors to obtain the result of Tucker decomposition. Although few methods have tried to reduce their computational time by sampling tensors, sketching tensors, and efficient matrix operations, their speed and memory efficiency are limited. In this paper, we propose D-Tucker, a fast and memory-efficient method for Tucker decomposition on large dense tensors. D-Tucker consists of the approximation, the initialization, and the iteration phases. D-Tucker 1) compresses an input tensor by computing randomized singular value decomposition of matrices sliced from the input tensor, and 2) efficiently obtains orthogonal factor matrices and a core tensor by using SVD results of sliced matrices. Through experiments, we show that D-Tucker is up to $38.4\times$ faster, and requires up to $17.2\times$ less space than existing methods with little sacrifice in accuracy.**

***Index Terms*—dense tensor, tucker decomposition, efficiency**

I. Introduction

How can we efficiently find latent patterns and relations in large dense tensors? Many real-world dense data, including brain image, spectral images, and air quality, are represented as multi-dimensional arrays, or tensors [1], [2]. Tucker decomposition [3], which factorizes a given tensor into factor matrices and a core tensor to find hidden concepts and latent patterns, has been widely used for analyzing tensors with various applications.

Alternating Least Square (ALS) is the most widely used method for Tucker decomposition. Existing ALS based methods, however, do not provide one or more of the desired properties for dense tensor decompositions: fast running time, efficient memory usage, and high accuracy. A few ALS based methods slightly reduce the computational time using efficient matrix operations [4] or applying randomized algorithms [5]. Moreover, other Tucker decomposition methods [6], [7] reduce the computational time and the memory requirement by approximating large dense tensors. However, none of them provides both fast running time with high accuracy and low memory usage. The major challenges to decompose dense tensors are 1) to minimize computational costs while giving low error, and 2) to avoid intermediate data explosion which leads to heavy computational costs and memory requirements.

In this paper, we propose D-Tucker, a fast and memory-efficient method for Tucker decomposition on large dense tensors. The main ideas of D-Tucker are: 1) to compress matrices sliced from an input tensor by exploiting randomized singular value decomposition (SVD), 2) to use the SVD results in initializing and updating factor matrices and a core tensor, and 3) to carefully determine ordering of computation for efficiency. D-Tucker consists of three phases: approximation, initialization, and iteration. In the approximation phase, D-Tucker computes randomized SVD [8] of matrices sliced from the input tensor, so that we reduce the size of the input tensor for updating the factor matrices and the core tensor. In the initialization phase, D-Tucker provides a starting point by computing orthogonal factor matrices using the SVD results of sliced matrices. Then, D-Tucker iteratively updates the factor matrices and the core tensor for reducing reconstruction error in the iteration phase by utilizing the SVD results. D-Tucker directly uses the SVD results, and avoids reconstruction of the approximated tensor for better time and space efficiency. Through comprehensive experiments, we show that D-Tucker is fast and memory-efficient compared to existing methods.

The contributions of this paper are as follows.

- **Algorithm.** We propose D-Tucker, a fast and memory-efficient method for decomposing dense tensors.
- **Experiment.** Extensive experiments show that D-Tucker is up to $38.4\times$ faster and requires up to $17.2\times$ less space than competitors (see Figure 1).

D-Tucker is available at https://datalab.snu.ac.kr/dtucker.

II. Related Work

We describe the related works for Tucker decomposition methods. De Lathauwer et al. [9] proposed Tucker-ALS which alternately updates factor matrices and obtains core tensor. Che et al. [5] applied randomized algorithms for Tucker decomposition. The main challenges of Tucker decomposition are heavy computational time and large memory requirements due to large-scale dense tensors; thus recent works [6], [7] approximate a large dense tensor with a small tensor, and updates the factor matrices and the core tensor using the small approximated tensor. Tsourakakis proposed MACH [6] method which randomly chooses elements of an input tensor with probability p. While MACH uses a sampled tensor in update phase, MACH has an accuracy issue due to the sampling. It also provokes heavy computational cost and memory requirement to update factor matrices and a core tensor at

each iteration. Malik et al. [7] proposed Tucker-ts which uses sketching in updating factor matrices and a core tensor. Tucker-ts generates small sketching tensors for each mode, and uses the sketching tensors in update phase. However, it requires a heavy computational cost to approximate a tensor since it performs sketching for all modes.

III. Proposed Method

A. Approximation Phase

The purpose of the approximation phase is to compress the input data with low error in order to increase memory efficiency and reduce the number of flops in the iteration phase. Our idea is to exploit two characteristics of a group of real-world tensors: 1) skewed shape, and 2) low dimensional structure in sliced matrices. We reorder modes of a given tensor based on the first characteristic, and compress the sliced matrices of the reordered tensor using a fast dimensionality reduction technique, randomized SVD.

Skewed shape of real-world tensors. Many real-world tensors have a skewed shape, where some of the modes have much smaller dimensionalities compared to those of other modes. We reorder modes in descending order of dimensionality and represent the reordered tensor as $\mathcal{X}_r \in \mathbb{R}^{I_1 \times I_2 \times K_3 \times \cdots \times K_N}$ where I_1 and I_2 are the two largest dimensionalities, K_n for $n = 3, 4, ..., N$ are the remaining dimensionalities, and $I_1 \geq I_2 \geq K_3 \geq \cdots \geq K_N$.

Low dimensional structure in sliced matrices. Sliced matrices of a given real-world tensor for any two modes often have a low dimensional structure which allows D-Tucker to compress the given tensor data with low errors. Furthermore, there is a computational benefit: by creating sliced matrices as opposed to sub-tensors, D-Tucker leverages the randomized SVD [10] with sparse embedding matrix [7], [11] to yield faster performance in the approximation phase. D-Tucker avoids working with tensor decomposition methods for sub-tensors, which leads to better efficiency.

We first express a reordered tensor $\mathcal{X}_r \in \mathbb{R}^{I_1 \times I_2 \times K_3 \times \cdots \times K_N}$ as a collection of sliced matrix $\mathbf{X}_{::k_3 \ldots k_N}$. We formally define the sliced matrix $\mathbf{X}_{::k_3 \ldots k_N}$ in Definition 1.

Definition 1 (Sliced matrix $\mathbf{X}_{::k_3 \ldots k_N}$)**.** *Given a reordered tensor $\mathcal{X}_r \in \mathbb{R}^{I_1 \times I_2 \times K_3 \times \cdots \times K_N}$, we extract sliced matrices by slicing the reordered tensor $\mathcal{X}_r$ so that the size of each sliced matrix $\mathbf{X}_{::k_3 \ldots k_N}$ is $I_1 \times I_2$.* □

After slicing the tensor $\mathcal{X}_r$ into the matrices $\mathbf{X}_{::k_3 \ldots k_N}$, we decompose the sliced matrix using randomized SVD [10] with sparse embedding matrix [7], [11].

$$\mathbf{X}_{::k_3 \ldots k_N} \simeq \mathbf{U}_{::k_3 \ldots k_N} \mathbf{\Sigma}_{::k_3 \ldots k_N} \mathbf{V}^{\mathrm{T}}_{::k_3 \ldots k_N} \tag{1}$$

where $\mathbf{U}_{::k_3 \ldots k_N}$ ($\in \mathbb{R}^{I_1 \times r}$) is a left singular vector matrix, $\mathbf{\Sigma}_{::k_3 \ldots k_N}$ ($\in \mathbb{R}^{r \times r}$) is a singular value matrix, $\mathbf{V}_{::k_3 \ldots k_N}$ ($\in \mathbb{R}^{I_2 \times r}$) is a right singular vector matrix, and the rank r is much smaller than the dimensionalities I_1 and I_2.

B. Initialization Phase

D-Tucker initializes factor matrices with Sequentially Truncated Higher-Order SVD (ST-HOSVD) [12] using the SVD results from the approximation phase; well-initialized factor matrices help reduce the number of iterations. Our idea is to initialize factor matrices without reconstructing $\mathcal{X}$ from the SVD results of sliced matrices, which reduces the computational cost and memory usage for the initialization. Note that we use standard SVD [13] for the effectiveness of the initialization.

First mode. We compute a factor matrix for the first mode. We represent mode-1 matricized matrix $\mathbf{X}_{(1)}$ of the reordered tensor $\mathcal{X}_r$ as follows:

$$\mathbf{X}_{(1)} = [\mathbf{X}_{::1,\ldots,1}; \cdots; \mathbf{X}_{::K_3,\ldots,K_N}] = [\mathbf{X}_1; \cdots; \mathbf{X}_l; \cdots; \mathbf{X}_L]$$

where L is equal to $K_3 \times \cdots \times K_N$, and the index l is defined as follows:

$$l = 1 + \sum_{i=3}^{N} \left((k_i - 1) \prod_{m=3}^{i-1} K_m \right)$$

where K_m is the dimensionality of mode-m, N is the order of the input tensor, and $\prod_{m=3}^{i-1} K_m$ is equal to 1 if $i - 1 < m$. Note that we represent a sliced matrix as $\mathbf{X}_l$ with the index l instead of $\mathbf{X}_{::k_3 \ldots k_N}$. Using the SVD of sliced matrices, the mode-1 matricized matrix $\mathbf{X}_{(1)}$ is expressed as follows:

$$\mathbf{X}_{(1)} = [\mathbf{X}_1; \cdots; \mathbf{X}_l; \cdots; \mathbf{X}_L] \simeq [\tilde{\mathbf{X}}_1; \cdots; \tilde{\mathbf{X}}_l; \cdots; \tilde{\mathbf{X}}_L] \tag{2}$$

where $\tilde{\mathbf{X}}_l$ is a representation of $\mathbf{U}_l \mathbf{\Sigma}_l \mathbf{V}_l^{\mathrm{T}}$. We carefully decouple $\mathbf{U}_l \mathbf{\Sigma}_l$ and $\mathbf{V}_l^{\mathrm{T}}$, and perform SVD of the concatenated matrix consisting of $\mathbf{U}_l \mathbf{\Sigma}_l$ for $l = 1..L$ [14], [15].

$$\begin{aligned} \mathbf{X}_{(1)} &\simeq [\mathbf{U}_1 \mathbf{\Sigma}_1; \cdots; \mathbf{U}_L \mathbf{\Sigma}_L] \times (blkdiag(\{\mathbf{V_1}\}_{l=1}^{L}))^{\mathbf{T}} \\ &\simeq \mathbf{U} \mathbf{\Sigma} \mathbf{V}^{\mathrm{T}} (blkdiag(\{\mathbf{V_1}\}_{l=1}^{L}))^{\mathbf{T}} \end{aligned} \tag{3}$$

where $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$ is the SVD result of the concatenated matrix $[\mathbf{U}_1 \mathbf{\Sigma}_1; \cdots; \mathbf{U}_L \mathbf{\Sigma}_L]$, $L = K_3 \times \cdots \times K_N$, and $blkdiag(\{\mathbf{V}_l\}_{l=1}^{L})$ is a block diagonal matrix consisting of $\{\mathbf{V}_l\}$. We obtain the initial factor matrix $\mathbf{A}^{(1)} = \mathbf{U}$.

Second mode. We compute a factor matrix for the second mode. We compute 1-mode product using the SVD results instead of the given tensor, and then perform SVD for the second mode. As shown in Equation (2), we matricize the tensor along mode-1 using the sliced matrices. Then, we perform matrix multiplication between the factor matrix $\mathbf{A}^{(1)\mathbf{T}}$ of the first mode and mode-1 matricized matrix as follows:

$$\begin{aligned} \mathbf{A}^{(1)\mathbf{T}} \mathbf{X}_{(1)} &\simeq \mathbf{A}^{(1)\mathbf{T}} [\mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^{\mathrm{T}}; \cdots; \mathbf{U}_L \mathbf{\Sigma}_L \mathbf{V}_L^{\mathrm{T}}] \\ &= \mathbf{Y}_{(2),inter} blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L}) \end{aligned} \tag{4}$$

where $\mathbf{Y}_{(2),inter} = \mathbf{A}^{(1)\mathbf{T}} [\mathbf{U}_1; \cdots; \mathbf{U}_L]$, $L = K_3 \times \cdots \times K_N$, and $blkdiag(\{\mathbf{\Sigma_l} \mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L})$ is a block diagonal matrix consisting of $\mathbf{\Sigma_l} \mathbf{V}_l^{\mathrm{T}}$. We compute Equation (4) by computing $\mathbf{Y}_{(2),inter}$ and multiplying it with the block diagonal matrix; we reshape the result of $\mathbf{Y}_{(2),inter} blkdiag(\{\mathbf{\Sigma}_l \ \mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L})$ as a tensor $\mathcal{Y}$ ($\in \mathbb{R}^{J_1 \times I_2 \times K_3 \times \cdots \times K_N}$). After the reshaping, we obtain the initial factor matrix $\mathbf{A}^{(2)}$ of the second mode by computing left singular vectors of mode-2 matricized matrix $\mathbf{Y}_{(2)}$.

Remaining modes. We observe that the mode-1 matricization of $\mathcal{X} \times_1 \mathbf{A}^{(1)\mathbf{T}} \times_2 \mathbf{A}^{(2)\mathbf{T}}$ is given by the following equation.

$$\begin{aligned} &\mathbf{A}^{(1)\mathbf{T}} \mathbf{X}_{(1)} blkdiag(\{\mathbf{A}^{(2)}\}_{l=1}^{L}) \\ &\simeq \mathbf{Y}_{(2),inter} blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l^{\mathrm{T}} \mathbf{A}^{(2)}\}_{l=1}^{L}) \end{aligned} \tag{5}$$

Note that $\mathbf{Y}_{(2),inter} = (\mathbf{A}^{(1)\mathbf{T}} [\mathbf{U}_1; \cdots; \mathbf{U}_L])$ has been already computed and saved as $\mathbf{Y}_{reuse}$ when computing the factor matrix of the second mode; thus we reuse $\mathbf{Y}_{reuse}$ when computing Equation (5). After computing mode-1 matricization

of $\mathcal{X} \times_1 \mathbf{A}^{(1)\mathbf{T}} \times_2 \mathbf{A}^{(2)\mathbf{T}}$, we reshape the result as a tensor $\mathcal{Y}$ ($\in \mathbb{R}^{J_1 \times J_2 \times K_3 \times \cdots \times K_N}$), perform SVD of $\mathbf{Y}_{(3)}$, and store $\mathcal{Y}$ as $\mathcal{Y}_{reuse}$ for remaining modes.

For modes $i = 4, 5, ..., N$, we perform $(i-1)$-mode product with $\mathcal{Y}_{reuse}$ stored for the previous mode-$(i-1)$ and initialize the factor matrix $\mathbf{A}^{(i)}$ as left singular vectors of mode-i matricization of $\mathcal{X} \times_1 \mathbf{A}^{(1)\mathbf{T}} \times_2 \mathbf{A}^{(2)\mathbf{T}} \cdots \times_{i-1} \mathbf{A}^{(i-1)\mathbf{T}}$.

C. Iteration Phase

The objective of the iteration phase is to alternately update factor matrices and compute the core tensor by efficiently computing n-mode products. Our ideas are to 1) avoid the reconstruction from the SVD results by exploiting their special structure, 2) carefully order matrix multiplications, and 3) avoid redundant computations for the first and second modes. Our ideas help D-Tucker avoid the rapid growth of computational time as the number of iterations increases. We use standard SVD [13] for stable convergence in this phase.

First mode. We update the first factor matrix $\mathbf{A}^{(1)}$ using the initialized factor matrices and the SVD results of the sliced matrices. We first compute 2-mode product using the SVD results instead of the given tensor, and then perform products for the remaining modes $3, 4, ..., N$. We matricize the tensor along mode-2 with the sliced matrices as follows:

$$\mathbf{X}_{(\mathbf{2})} = [\mathbf{X}_1^{\mathrm{T}}; \cdots; \mathbf{X}_l^{\mathrm{T}}; \cdots; \mathbf{X}_L^{\mathrm{T}}] \simeq [\tilde{\mathbf{X}}_1^{\mathbf{T}}; \cdots; \tilde{\mathbf{X}}_l^{\mathbf{T}}; \cdots; \tilde{\mathbf{X}}_L^{\mathbf{T}}]$$

Then, we perform matrix multiplication between the factor matrix $\mathbf{A}^{(2)\mathbf{T}}$ of the second mode and the mode-2 matricized matrix as follows:

$$\begin{aligned} &\mathbf{A}^{(2)\mathbf{T}}\mathbf{X}_{(\mathbf{2})} \simeq \mathbf{A}^{(2)\mathbf{T}} [\mathbf{V}_1 \mathbf{\Sigma}_1 \mathbf{U}_1^{\mathrm{T}}; \cdots; \mathbf{V}_L \mathbf{\Sigma}_L \mathbf{U}_L^{\mathrm{T}}] \\ &= \mathbf{Y}_{(1),inter} blkdiag(\{\mathbf{\Sigma}_l \mathbf{U}_l^{\mathrm{T}}\}_{l=1}^{L}) \end{aligned} \tag{6}$$

where $\mathbf{Y}_{(1),inter} = \mathbf{A}^{(2)\mathbf{T}} \left[\mathbf{V}_1; \cdots; \mathbf{V}_L\right]$, $L = K_3 \times \cdots \times K_N$, and $blkdiag(\{\mathbf{\Sigma}_l \mathbf{U}_l^{\mathrm{T}}\}_{l=1}^{L})$ is a block diagonal matrix consisting of $\mathbf{\Sigma}_l \mathbf{U}_l^{\mathrm{T}}$. We compute Equation (6) by computing $\mathbf{Y}_{(1),inter}$, and multiplying it with the block diagonal matrix; then, we reshape the result of $\mathbf{Y}_{(1),inter} blkdiag(\{\mathbf{\Sigma}_l\ \mathbf{U}_l^{\mathrm{T}}\}_{l=1}^{L})$ as $\mathcal{Y}$ ($\in \mathbb{R}^{I_1 \times J_2 \times K_3 \times \cdots \times K_N}$). After computing Equation (6), we perform the remaining n-mode products with $\mathcal{Y}$ for $n = 3, 4, ..., N$, and then update the factor matrix $\mathbf{A}^{(1)}$ by computing SVD of mode-1 matricized matrix $\mathbf{Y}_{(1)}$.

Second mode. We update the second factor matrix $\mathbf{A}^{(2)}$ by computing 1-mode product using the SVD results and then performing products for the remaining modes $3, 4, ..., N$. As shown in Equation (2), we matricize the tensor along mode-1 with the sliced matrices. Then, we perform matrix multiplication between the factor matrix $\mathbf{A}^{(1)\mathbf{T}}$ of the first mode and the mode-1 matricized matrix as in Equation (4). We first compute $\mathbf{Y}_{(2),inter} = \mathbf{A}^{(1)\mathbf{T}} \left[\mathbf{U}_1; \cdots; \mathbf{U}_L\right]$, multiply it with the block diagonal matrix $blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L})$, and then reshape the result of $\mathbf{Y}_{(2),inter} blkdiag(\{\mathbf{\Sigma}_l\ \mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L})$ as $\mathcal{Y}$ ($\in \mathbb{R}^{J_1 \times I_2 \times K_3 \times \cdots \times K_N}$). Note that we store $\mathbf{Y}_{(2),inter}$ to reuse it when computing the factor matrices $\mathbf{A}^{(i)}$ for $i = 3, 4, ..., N$ and the core tensor. After the reshaping, we perform the remaining n-mode products with $\mathcal{Y}$ for $n = 3, 4, ..., N$, and then update the factor matrix $\mathbf{A}^{(2)}$ by computing SVD of mode-2 matricized matrix $\mathbf{Y}_{(2)}$.

TABLE I
DESCRIPTION OF REAL-WORLD TENSOR DATASETS.

Dataset	Order	Dimensionality	Target rank
Brainq [16]	3	$(360, 21764, 9)$	$(10, 10, 5)$
Boats [17]	3	$(320, 240, 7000)$	$(10, 10, 10)$
Air Quality[1]	3	$(30648, 376, 6)$	$(10, 10, 5)$
HSI [18]	4	$(1021, 1340, 33, 8)$	$(10, 10, 10, 5)$

Remaining modes and core tensor. We update factor matrices $\mathbf{A}^{(i)}$ for all $i = 3, 4, ..., N$, and the core tensor $\mathcal{G}$. The mode-1 matricization of $\mathcal{X} \times_1 \mathbf{A}^{(1)\mathbf{T}} \times_2 \mathbf{A}^{(2)\mathbf{T}}$ is given by Equation (5). We avoid redundant computation by reusing the saved $\mathbf{Y}_{(2),inter}$. Note that reusing just $\mathbf{Y}_{(2),inter}$ reduces computational costs since $\mathbf{Y}_{(2),inter}$ is much smaller than the input tensor $\mathcal{X}$ and the SVD results of sliced matrices. After computing $\mathbf{Y}_{(2),inter} blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l^{\mathrm{T}} \mathbf{A}^{(2)}\}_{l=1}^{L})$ and reshaping the result as $\mathcal{Y}_{reuse}$ ($\in \mathbb{R}^{J_1 \times J_2 \times K_3 \cdots \times K_N}$) once, the reshaped tensor is reused to compute factor matrices $\mathbf{A}^{(i)}$ for $i = 3, 4, ..., N$ and the core tensor $\mathcal{G}$. The factor matrix $\mathbf{A}^{(i)}$ is updated by performing the remaining n-mode products, and SVD of $\mathbf{Y}_{(i)}$. For the core tensor, we perform n-mode products between the reshaped tensor $\mathcal{Y}_{reuse}$ ($\in \mathbb{R}^{J_1 \times J_2 \times K_3 \cdots \times K_N}$) and $\mathbf{A}^{(n)\mathbf{T}}$ for all $n = 3, 4, ..., N$.

IV. EXPERIMENT

A. Experimental Settings

We use a workstation with a single CPU (Intel Xeon E5-2630 v4 @ 2.2GHz) and 512GB memory. We use four real-world dense tensors described in Table I.

Competitors. We compare D-Tucker with Tucker-ALS, MACH [6], Randomized Tucker Decomposition (RTD) [5], Tucker-ts [7], and Tucker-ttmts [7]. All the methods including D-Tucker are implemented in MATLAB (R2019b).

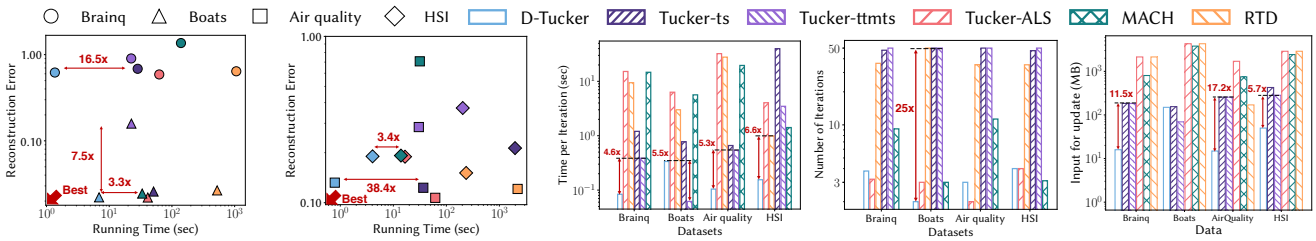
Parameters. We use a single thread. The maximum number of iterations is set to 50. We set the dimensionality J_n of the nth mode of a core tensor to 10, but we set it to 5 when the dimensionality K_n of the nth mode of an input tensor is smaller than 10. We also set the rank J of randomized SVD to 10. The iteration stops when the variation of the error $\frac{\sqrt{\|\mathcal{X}\|_{\mathbf{F}}^2 - \|\mathcal{G}\|_{\mathbf{F}}^2}}{\|\mathcal{X}\|_{\mathbf{F}}}$ [19] is less than $\epsilon = 10^{-4}$.

Reconstruction error. We evaluate the accuracy in terms of reconstruction error $\frac{\|\mathcal{X} - \hat{\mathcal{X}}\|_{\mathbf{F}}^2}{\|\mathcal{X}\|_{\mathbf{F}}^2}$ where $\mathcal{X}$ is an input tensor and $\hat{\mathcal{X}}$ is the reconstruction from the output of Tucker decomposition.

B. Performance

Time and reconstruction error. We evaluate the running time and the reconstruction error of D-Tucker compared to other Tucker decomposition methods based on ALS. Figures 1(a) and 1(b) show D-Tucker outperforms competitors in terms of the running time to obtain factor matrices and core tensor, while giving a reconstruction error close to that of the most accurate method Tucker-ALS. D-Tucker is up to $38.4\times$ faster than Tucker-ts, Tucker-ttmts, and MACH with smaller or similar reconstruction errors. Although Tucker-ALS and RTD have smaller errors for Air quality and HSI datasets, they are at least $3.4\times$ and $42\times$ slower than D-Tucker, respectively.

[1] https://www.airkorea.or.kr



(a) Run. time vs. error for Brainq and Boats datasets

(b) Run. time vs. error for Air quality and HSI datasets

(c) Time per iteration

(d) Number of iterations

(e) Memory Usage

Fig. 1. D-Tucker provides the best performance in terms of error, running time, and memory usage. (a) (b) D-Tucker provides the best tradeoff between running time and error. (c) Space cost of D-Tucker. (d) The running time of each iteration of D-Tucker. (e) The number of iterations of D-Tucker.

Efficiency of the iteration phase. We compare the running time per iteration and the number of iterations of D-Tucker to those of ALS based competitors in Figures 1(c) and 1(d). Contrary to other methods, D-Tucker avoids rapid growth of the total running time as the number of iterations increases; the overall running time of D-Tucker is not proportional to the running time per iteration since the approximation phase of D-Tucker is dominant. In Figure 1(c), the running time of each iteration of D-Tucker is at least $4.6\times$ lower than those of other competitors except for Boats dataset. For Boats dataset, the running time of each iteration of Tucker-ttmts is lower than that of D-Tucker, but Tucker-ttmts requires larger number of iterations than D-Tucker, and thus D-Tucker is $4.5\times$ faster than Tucker-ttmts at the iteration phase. Figure 1(d) shows that the number of iterations of D-Tucker is smaller than those of all competitors except Tucker-ALS, for 3-order datasets; the number of iterations of D-Tucker is up to $1.5\times$ larger than that of Tucker-ALS, but the difference is quite small considering the running time per iteration. For the 4-order dataset, the number of iterations of D-Tucker is $1.3\times$ larger than that of MACH, but the running time per iteration of D-Tucker is $6.6\times$ lower than that of MACH.

Space cost. We compare the memory usage of methods for initializing and updating factor matrices and core tensor. Figure 1(e) shows D-Tucker outperforms competitors in terms of memory usage to obtain factor matrices and core tensor. D-Tucker requires up to $17.2\times$ smaller space than Tucker-ts and Tucker-ttmts which are the second best methods in terms of space usage. For Boats dataset, Tucker-ts and Tucker-ttmts are efficient in terms of space since Boats dataset has a setting (order N and rank J are very small, and dimensionalities I and K are very large) where Tucker-ts and Tucker-ttmts operate well. Still, D-Tucker requires $1.03\times$ less space than Tucker-ts. Although D-Tucker requires $2.1\times$ more space than Tucker-ttmts, D-Tucker achieves $7.5\times$ less error than Tucker-ttmts.

V. Conclusions

We propose D-Tucker, a fast and memory-efficient Tucker decomposition method for large-scale dense tensors. D-Tucker compresses an input tensor by performing randomized SVD of matrices sliced from the tensor, and efficiently obtains factor matrices and a core tensor by 1) avoiding reconstruction from the compressed data, and 2) careful ordering of computations. Extensive experiments show that D-Tucker is up to $38.4\times$ faster, and requires up to $17.2\times$ less space than existing methods with little sacrifice in accuracy. Future works include extending the method for parallel and distributed systems.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) funded by MSIT(2019R1A2C2004990). U Kang is the corresponding author.

References

[1] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *KDD*, 2012, pp. 316–324.

[2] D. Choi, J.-G. Jang, and U. Kang, "S3cmtf: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization," *PLOS ONE*, vol. 14, no. 6, pp. 1–20, 06 2019.

[3] S. Oh, N. Park, J. Jang, L. Sael, and U. Kang, "High-performance tucker factorization on heterogeneous platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2237–2248, 2019.

[4] B. W. Bader and T. G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping," *ACM Transactions on Mathematical Software*, vol. 32, no. 4, pp. 635–653, Dec. 2006.

[5] M. Che and Y. Wei, "Randomized algorithms for the approximations of tucker and the tensor train decompositions," *Adv. Comput. Math.*, vol. 45, no. 1, pp. 395–428, 2019.

[6] C. E. Tsourakakis, "MACH: fast randomized tensor decompositions," in *SDM*, 2010, pp. 689–700.

[7] O. A. Malik and S. Becker, "Low-rank tucker decomposition of large tensors using tensorsketch," in *NeurIPS*, 2018, pp. 10 117–10 127.

[8] N. Halko, P. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[9] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "On the best rank-1 and rank-(R_1 , R_2, ... , R_N) approximation of higher-order tensors," *SIMAX*, vol. 21, no. 4, pp. 1324–1342, 2000.

[10] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, "A fast randomized algorithm for the approximation of matrices," *ACHA*, 2008.

[11] K. L. Clarkson and D. P. Woodruff, "Low-rank approximation and regression in input sparsity time," *JACM*, vol. 63, no. 6, p. 54, 2017.

[12] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SISC*, vol. 34, no. 2, 2012.

[13] J. Baglama and L. Reichel, "Augmented implicitly restarted lanczos bidiagonalization methods," *SISC*, vol. 27, no. 1, pp. 19–42, 2005.

[14] M. Iwen and B. Ong, "A distributed and incremental svd algorithm for agglomerative data analysis on large networks," *SIMAX*, 2016.

[15] J.-G. Jang, D. Choi, J. Jung, and U. Kang, "Zoom-svd: Fast and memory efficient method for extracting key patterns in an arbitrary time range," in *CIKM*. ACM, 2018, pp. 1083–1092.

[16] T. M. Mitchell, S. V. Shinkareva, A. Carlson, K.-M. Chang, V. L. Malave, R. A. Mason, and M. A. Just, "Predicting human brain activity associated with the meanings of nouns," *Science*, 2008.

[17] Y. Wang, P. Jodoin, F. M. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, "Cdnet 2014: An expanded change detection benchmark dataset," in *CVPR Workshops*, 2014, pp. 393–400.

[18] D. Foster, K. Amano, S. Nascimento, and M. Foster, "Frequency of metamerism in natural scenes," *Optical Society of America. Journal A: Optics, Image Science, and Vision*, 2006.

[19] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.