VEST: Very Sparse Tucker Factorization of Large-Scale Tensors

Moonjeong Park*
Pohang University of Science and Technology
Pohang, Republic of Korea
mjeongp@postech.ac.kr

Jun-Gi Jang*
Seoul National University
Seoul, Republic of Korea
elnino4@snu.ac.kr

Lee Sael†
Ajou University
Suwon, Republic of Korea
sael@ajou.ac.kr

***Abstract*—Given a large tensor, how can we decompose it to sparse core tensor and factor matrices without reducing the accuracy? Existing approaches either output dense results or have scalability issues. In this paper, we propose VEST, a tensor factorization method for large partially observable data to output a very sparse core tensor and factor matrices. VEST performs initial decomposition and iteratively determines unimportant entries in the decomposition results, removes the unimportant entries, and updates the remaining entries. To determine unimportant entries of factor matrices and core tensor, we define and use the entry-wise 'responsibility' of the current decomposition. For scalable computation, the entries are updated iteratively using a carefully derived coordinate descent rule in parallel. Also, VEST automatically searches for the best sparsity ratio that results in a balanced trade-off between sparsity and accuracy. Extensive experiments show that our method VEST produces more accurate results compared to the best performing competitors for all tested real-life datasets.**

***Index Terms*—Scalable tensor factorization, Tucker, Sparsity**

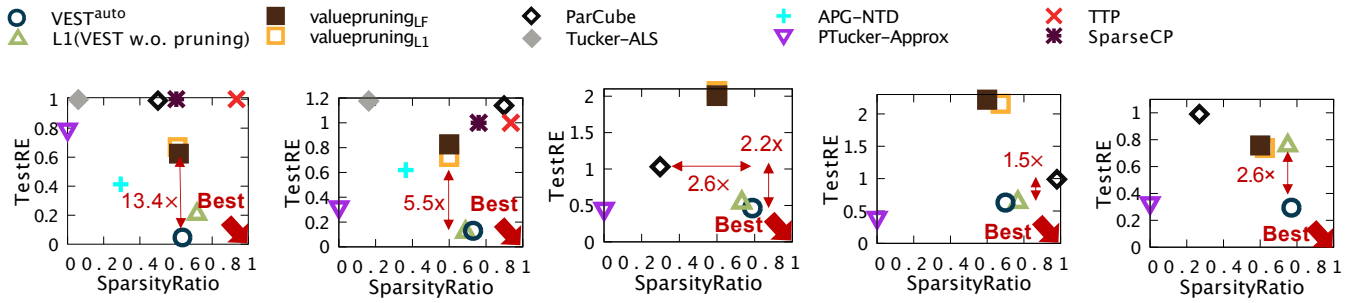
I. INTRODUCTION

How can we factorize a large tensor to sparse core tensor and factor matrices without sacrificing accuracy and apply them on partially observable tensors? A tensor is a powerful tool for representing multi-modal data. Analysis of tensors often evolves tensor factorization. A Tucker factorization is widely used tensor factorization form that outputs a core tensor and factor matrices which reveal the latent relation of the data. Tucker factorization can also be viewed as a tool for multi-linear regression problem where only the target values, i.e., values of input tensor entries, are known. In this perspective, the columns of factor matrices act as latent components, their values as latent feature values, and the cells of the core tensor as weights of the relations between the latent components [1]. Sparse Tucker factorization aims to output sparse core tensor and factor matrices for better interpretability. As sparse linear regression model enhances its interpretability [2], sparse factor matrices and a core improve interpretability. Sparsity of results is well known to have various advantages such as improving interpretability and memory usage. In particular, the effect of sparsity in improving interpretability has been supported and exploited by various studies [2]–[9]. Furthermore, real data are full of missing data (partially observable) and analysis methods should have strategies to address the missing data problem.

There are several approaches for sparsifying Tucker factorization results. The widely used general approach adds an L_1 norm as sparse regularizer in their objective function [6], [7], [10], [11]. However, the sparsity of a L_1-based factorization results is sensitive to the lambda values, in which extensive search of appropriate lambda value is required to obtain requested sparsity. Moreover, in some cases, it is difficult to obtain a very sparse result with even a large lambda value. Another general approach removes elements with small values from the core tensor or the factor matrices [3], [8], [9]. However, removing such elements does not necessarily lead to small reconstruction errors, and thus value-based pruning sacrifices accuracy. In addition to the two sparsification strategies, CANDECOMP/PARAFAC (CP) decomposition can also be considered as a sparse method. More specifically, a CP decomposition can be considered as a specific type of Tucker factorization that generates a very sparse core tensor, i.e. a core tensor with non-zero values only at the super-diagonal positions. However, CP does not generate sparse factor matrices and requires additional steps for sparcification [3], [12] There are also, application-specific approaches that rely on application-specific assumptions that are not generalizable. Example of application-specific methods include utilizing domain-specific knowledge as sparsity constraints [5], constructing factor matrix from sparse input sampling [4], [13], using smoothing matrices [14], [15], and learning sparse dictionary for image data [16]–[18].

In this paper, we propose VEST (VEry Sparse Tucker factorization), a scalable and accurate Tucker factorization method to generate sparse factors and a core tensor for large-scale partially observable input tensor. VEST outputs very sparse factorization results by carefully determining the importance of elements of factors and the core and pruning unimportant ones up till a stopping criterion that determines a reasonable sparsity-accuracy trade-off. VEST guarantees that the sparsity does not decrease in the update process by carefully derived update rules. The entries of factor matrices and core tensor are updated iteratively using our derived coordinate descent rules in parallel for scalable computation. The code of our proposed method is available at https://github.com/vestbigcomp/vestbigcomp2020/tree/master.

* These authors contributed equally.
† Corresponding author



(a) Yelp-s (b) Amazon-s (c) Yelp (d) Amazon (e) Movie

Figure 1. Sparsity and accuracy of VEST and competitors on 2-small and 3 large-scale real-world datasets. VEST generates sparse and accurate results that generalize well on unseen data: points of VEST are located in the bottom right region (the best point) of Test RE plots. Compared to competitors, i.e, ParCube, APG-NTD, PTucker-Approx, TTp, and Sparse CP, VEST is at least 2.6 times sparser and at least 2.2 times more accurate, i.e. (c) VEST compared to ParCube. Also, compared to different sparsification strategies , VEST provides lower TestRE and RE while having similar sparsity.

II. Preliminaries and Related Works

We introduce concepts of tensor and its operations, Tucker factorization, and the standard algorithm for Tucker. Table II lists the symbols used.

Table I
TABLE OF SYMBOLS AND DEFINITIONS.

Symbol	Definition
$\mathcal{X}$	input tensor ($\in \mathbb{R}^{I_1 \times \ldots \times I_N}$)
$\mathcal{G}$	core tensor ($\in \mathbb{R}^{J_1 \times \ldots \times J_N}$)
N	order of $\mathcal{X}$
I_n, J_n	dimensionality of the nth mode of $\mathcal{X}$ and $\mathcal{G}$, respectively
$\mathbf{A}^{(n)}$	nth factor matrix ($\in \mathbb{R}^{I_n \times J_n}$)
$a^{(n)}_{i_n j_n}$	(i_n, j_n)th element of $\mathbf{A}^{(n)}$
Ω	set of observable entries of $\mathcal{X}$
$\|\Omega\|$	number of observable entries of $\mathcal{X}$
$\Omega^{(n)}_{i_n}$	set of observable entries whose nth mode index is i_n
λ	regularization parameter for core and factor matrices
$\|\mathcal{X}\|_F$	Frobenius norm of tensor $\mathcal{X}$
$\|\mathcal{X}\|_1$	sum of absolute values of tensor $\mathcal{X}$
α	an entry $(i_1, ..., i_N)$ of input tensor $\mathcal{X}$
β	an element $(j_1, ..., j_N)$ of core tensor $\mathcal{G}$
$\alpha_{i_n=i}$	an entry $(i_1, ..., i_n = i, ..., i_N)$ of input tensor $\mathcal{X}$
$\beta_{j_n=j}$	an element $(j_1, ..., j_n = j, ..., j_N)$ of core tensor $\mathcal{G}$

A. Tensor and its Operations

Tensor is a multi-dimensional array that contains numbers. An 'order' or 'mode' is the number of tensor dimensions, where a 1^{st}-order tensor represents a vector and a 2^{nd}-order tensor represents a matrix. We denote vectors by boldface lowercase letters (e.g., $\mathbf{a}$), matrices by boldface capital letters (e.g., $\mathbf{A}$), and three or higher order tensors by boldface Euler script letters (e.g., $\mathcal{X}$). An entry of a 3^{rd}-order tensor can be expressed with three indices. For example, the (i_1, i_2, i_3)th entry of a 3^{rd}-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is denoted by $x_{i_1 i_2 i_3}$, where index i_n spans from 1 to I_n.

The size of a tensor is often evaluated by the Frobenius norm. Given an N-order tensor $\mathcal{X}$ ($\in \mathbb{R}^{I_1 \times \ldots \times I_N}$), the *Frobenius norm* of $\mathcal{X}$ is $||\mathcal{X}||_F = \sqrt{\sum_{\forall \alpha \in \mathcal{X}} \mathcal{X}^2_\alpha}$, where $\alpha = (i_1, \cdots, i_N)$ is an index to an entry of input tensor $\mathcal{X}$. Tensor decomposition often involves matricizations of tensors and products between a tensor and a matrix. The *mode-n matricization* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is denoted as $\mathbf{X}_{(n)}$ and the mapping from an entry $(i_1, \cdots, i_N)$ of $\mathcal{X}$ to an entry (i_n, j) of $\mathbf{X}_{(n)}$ is given by $j = 1 + \sum_{k=1, k \neq n}^{N} [(i_k - 1) \prod_{m=1, m \neq n}^{k-1} I_m]$. Also, the *n-mode product* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$ ($\in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$). Entrywise, *n-mode product* is denoted as $(\mathcal{X} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i=1}^{I_n} (\mathcal{X}_{\alpha_{i_n=i}} u_{ji})$.

B. Tucker Factorization

Our proposed method VEST is built on top of Tucker factorization, one of the most popular tensor factorization methods. Given an N^{th}-order tensor $\mathcal{X}$ ($\in \mathbb{R}^{I_1 \times \cdots \times I_N}$), Tucker factorization approximates $\mathcal{X}$ by a core tensor $\mathcal{G}$ ($\in \mathbb{R}^{J_1 \times \cdots \times J_N}$) and factor matrices $\{\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} | n = 1 \cdots N\}$ by minimizing the full reconstruction error:

$$L(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \|\mathcal{X} - \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}\|_F. \quad (1)$$

Typically, a core tensor $\mathcal{G}$ is assumed to be smaller and denser than the input tensor $\mathcal{X}$. Each factor matrix $\mathbf{A}^{(n)}$ represents the latent features of the object related to the nth mode of $\mathcal{X}$, and each element of a core tensor $\mathcal{G}$ indicates the weights of the relations composed of columns of factor matrices.

1) Sparsification Strategies: Tucker factorization often results in dense core and factor matrices. One of the approaches for sparsifying results is by including a sparsity constraint in the form of L_1 norm, a.k.a., Lasso, into the objective function (e.g., L = Eq. (1)).

$$L_1(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = L + \lambda(\|\mathcal{G}\|_1 + \sum_{n=1}^{N} \|\mathbf{A}^{(n)}\|_1)$$

Another approach for sparsifying results is by pruning minimal elements. That is the *value pruning* method sets the smallest s ratio of the elements to zero in the core and factor matrices.

2) Addressing Missing Values: In real-world, data are often incomplete with several missing entries. To accommodate for the missing data, we have previously proposed and validated

a partially observable Tucker factorization objective function [19]–[23].

Given a tensor $\mathcal{X}$ ($\in \mathbb{R}^{I_1 \times \cdots \times I_N}$) with observable entries Ω, the goal of *partially observable Tucker factorization* of $\mathcal{X}$, in combination with the L_1 regularization, is to find factor matrices $\mathbf{A}^{(n)}$ ($\in \mathbb{R}^{I_n \times J_n}, n = 1, \cdots, N$) and a core tensor $\mathcal{G}$ ($\in \mathbb{R}^{J_1 \times \cdots \times J_N}$) minimizing the following loss:

Given a tensor $\mathcal{X}$ with observable entries Ω, the goal of *partially observable Tucker factorization via Lasso* of $\mathcal{X}$ ($L1$ method in Fig. 1) is to find factor matrices and a core tensor that minimizing the following loss:

$$L_1(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \sum_{\forall \alpha \in \Omega} \left(\mathcal{X}_\alpha - \sum_{\forall \beta \in \mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \right)^2 + \lambda(\|\mathcal{G}\|_1 + \sum_{n=1}^{N} \|\mathbf{A}^{(n)}\|_1) \tag{2}$$

Again the reconstruction error in Eq. (2) depends only on the observable entries of $\mathcal{X}$, and L_1 regularization is used to enforce sparsity.

3) Reconstruction of Tucker Tensor: Evaluation of tensor decomposition and the prediction of the missing entry values (a.k.a., tensor completion) involves reconstruction. Given core tensor $\mathcal{G}$ and factor matrices $\mathbf{A}^{(n)}$, the *reconstruction* of the original tensor $\mathcal{X}$ is defined as $\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}$.

III. Proposed Method

In this section, we propose VEST (Very Sparse Tucker factorization), a method for partially observed large scale tensor that results in a very sparse core tensor and very sparse factor matrices. Sparsity of the results improves interpretability and provides a scheme for better compression. To maximize sparsity without losing accuracy, VEST iteratively updates core tensor and factor matrices, and prunes unimportant elements from the core tensor and factor matrices. However, there are several challenges in designing an efficient sparse update and pruning rules.

- **Evaluating importance of elements.** Vital elements of core tensor and factor matrices should not be pruned. How can we evaluate their importance?
- **Automatically determining the sparsity.** There is a trade-off relationship between sparsity and accuracy. How can we automatically determine an appropriate sparsity which gives a good balance with regards to accuracy?
- **Updating factors while guaranteeing non-decreasing sparsity.** The update process of factors and the core in the regular Tucker-ALS does not guarantee that the sparsity improves over the update process. How can we guarantee that update rules improve the sparsity while avoiding the stability problem?

We have the following main ideas to address the above challenges which we describe in detail in later subsections.

- **Design responsibility indicator** to evaluate contribution of each element on the accuracy (Section III-B).
- **Design auto-search algorithm** to find a good sparsity that resides near the maximum sparsity just before the reconstruction error shoots up (Section III-C).
- **Design element-wise update rules with slowly growing sparsity strategy** to independently update each element of factor matrices and the core tensor while addressing the stability problem. Element-wise update rules guarantee non-decreases of the sparsity by keeping pruned elements to zeros while slowly growing sparsity strategy alleviates the stability problem (Sections III-D and III-E).

A. Overview

Algorithm 1: VEST: Very Sparse Tucker Factorization

Input : Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, core tensor dimensionality $J_1, ..., J_N$, and target sparsity s (if manual-mode).

Output: Sparse factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$ and sparse core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$.

1 randomly initialize $\mathbf{A}^{(n)} (n = 1, \cdots, N)$ and $\mathcal{G}$; set pr = INIT_PR, $iterN$ = 0.
2 **repeat**
3 update unpruned elements of $\mathbf{A}^{(n)} (n = 1, \cdots, N)$ ▷ Algorithm 3
4 update unpruned elements of $\mathcal{G}$ ▷ (19)
5 compute RE using observable entries Ω ▷ Eq.(5)
6 **if** *should_prune()* **then**
7 prune pr=min(INIT_PR*$iterN$, MAX_PR) ratio of elements e in $\mathbf{A}^{(n)}$ and $\mathcal{G}$ based on $Resp(e)$ of the elements ▷ Algorithm 2
8 **until** *RE converges or iterN++ exceeds maximum iteration*;
9 **for** *n = 1...N* **do**
10 $\mathbf{U}^{(n)}\mathbf{B}^{(n)} \leftarrow \mathbf{A}^{(n)}$, and set $\mathbf{A}^{(n)} \leftarrow \mathbf{U}^{(n)}$
11 $\mathcal{G} \leftarrow \mathcal{G} \times_n \mathbf{B}^{(n)}$

VEST is a scalable Tucker factorization method for partially observable tensors that results in a very sparse core tensor and very sparse factor matrices (see Algorithm 1). First, VEST initializes all elements of the core tensor and factor matrices with random real values between 0 and 1 (line 1). Next, VEST iteratively updates the core tensor and factor matrices while pruning their elements (lines 3-7). In lines 3-4, VEST updates unpruned elements of the core tensor and factor matrices by element-wise update rules (Section III-D), guaranteeing that the sparsity non-decreases. Then VEST prunes unimportant elements in the core tensor and factor matrices (lines 6-7). Importance of each element e is evaluated by responsibility $Resp(e)$ which indicates how largely the element contributes to the accuracy (Section III-B). $should\_prune()$ function determines when to stop pruning: if desired sparsity s is achieved (in the manual version VEST^{man}) or the reconstruction error shows a rapid increase (VEST^{auto}, i.e., default VEST).

Motivated from simulated annealing, we start with minimal pruning rates gradually increase the pruning rate as iterations proceed (line 7); this enables to explore larger search space in the beginning, while reducing the extent of the search to reduce to a minimum in the later iterations such that instability of pruning on reconstruction error is alleviated. However, in the real-world datasets the hyparameters $INIT_{PR}/MAX_{PR}$ did not notably change the sparsity or the accuracy. The iterations proceed until the reconstruction error converges or the maximum iteration is reached. Finally, VEST standardizes all columns of factor matrices such that their norm is equal to one and updates core tensor accordingly (lines 9-11). Specifically, $\mathbf{A}^{(n)}$ is decomposed to $\mathbf{U^{(n)}} \in \mathbb{R}^{I_n \times J_n}$ and $\mathbf{B^{(n)}}$ where columns of $\mathbf{U^{(n)}}$ are unit vectors and $\mathbf{B^{(n)}}$ is a diagonal matrix whose (i,i)th element is the norm of $\mathbf{A}^{(n)}$'s ith column. The core tensors are updated to maintain the same reconstruction error [24].

B. Evaluating Importance of Elements by Responsibility

In this section, we evaluate the importance of elements of factor matrices and core tensor for pruning. A naive approach considers elements of factor matrices and core tensor which have small values as *unimportant* elements. However, pruning of a small valued element could provoke large reconstruction error when its overall affect is large (an example provided in the Supplementary [25]). Therefore, VEST calculates the *responsibility* of each element of factor matrices and core tensor by considering the overall affect of an element in the reconstruction. That is, responsibility represents its contribution to the overall reconstruction accuracy over the observable elements of the input tensor to determine and prune unimportant elements. The intuition is that reconstruction error increases significantly when a vital element of the core tensor or factor matrices is set to zero, i.e., pruned. On the other hand, if the reconstruction error after pruning is similar or even smaller to that before pruning, the pruned element is insignificant. Formally, the responsibility is defined as follows.

Definition 1 (Responsibility) Responsibility of an element e in a factor matrix ($e = a^{(n)}_{ij}$) or core tensor ($e = \mathcal{G}_\gamma$) is given by

$$Resp(e) = \frac{RE(e) - RE}{RE}, \tag{3}$$

where

$$RE = \frac{\sqrt{\sum_{\forall \alpha=(i_1,\cdots,i_N)\in\Omega} \left(\mathcal{X}_\alpha - \sum_{\forall \beta=(j_1,\cdots,j_N)\in\mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2}}{||\mathcal{X}||_F} \tag{5}$$

is the normalized reconstruction error over the observable entries Ω of the original tensor $\mathcal{X}$, and $RE(e)$ is residual reconstruction error defined when the element e is set to zero (Eq. (7) and (9)). □

Definition 2 (Residual reconstruction error) The residual reconstruction error $RE(\mathcal{G}_\gamma)$ for $(j_1, ..., j_N)$th element γ in core tensor $\mathcal{G}$ is as follows:

$$(RE(\mathcal{G}_\gamma))^2 = \frac{\sum_{\forall \alpha=(i_1,\cdots,i_N)\in\Omega} \left(\mathcal{X}_\alpha - \sum_{\forall \beta=(j_1,\cdots,j_N)\neq\gamma\in\mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}\right)^2}{||\mathcal{X}||^2_F} \tag{7}$$

The residual reconstruction error $RE(a^{(n)}_{i,j})$ for an (i,j)th element $a^{(n)}_{i,j}$ in a factor matrix $\mathbf{A}^{(n)}$ is as follows:

$$(RE(a^{(n)}_{ij}))^2 = RE^2 + \frac{\sum_{\forall \alpha\in\Omega^{(n)}_{i_n}} \left(2\cdot(\mathcal{X}_\alpha - B(\alpha)) + B_{j_n=j}(\alpha)\right)\cdot(B_{j_n=j}(\alpha))}{||\mathcal{X}||^2_F}, \tag{9}$$

where $B(\alpha)$ is the *entry-wise reconstruction* defined as

$$\mathcal{X}_{\alpha=(i_1,\cdots,i_N)} \approx B(\alpha) = \sum_{\forall \beta=(j_1,\cdots,j_N)\in\mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}, \tag{10}$$

and $B_{j_n=j}(\alpha)$ and $B_{j_n\neq j}(\alpha)$ are the *partial reconstruction functions* defined as

$$\begin{aligned} B_{j_n=j}(\alpha) &= \sum_{\forall \beta_{j_n=j}} \mathcal{G}_{\beta_{j_n=j}} \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \\ B_{j_n\neq j}(\alpha) &= \sum_{\forall \beta_{j_n\neq j}} \mathcal{G}_{\beta_{j_n\neq j}} \prod_{n=1}^{N} a^{(n)}_{i_n j_n}. \end{aligned} \tag{11}$$

□

The proof of correctness for the derivation of the Eq.(9) is provided in the Supplementary [25]. Note that both definitions are derived from the element-wise reformulation of the reconstruction error.

C. Pruning

VEST obtains sparse factor matrices and core tensor by a novel pruning method. After calculation of responsibility values, VEST prunes core tensor and factor matrices, i.e, the elements of factor matrices and core tensor that have small responsibility values are set to zero. Pruning technique allows VEST to be insensitive to the L_1 regularization parameter λ while still generating sparse factor matrices and core tensor.

Pruning is performed iteratively, each time after the core tensor and factor matrices are updated. The process of pruning an element consists of setting the value of the element to zero and marking the element as pruned in a marking table. The marked elements are excluded from the update step. To prune elements with low responsibility values, VEST sorts elements of the core tensor and each factor matrix, respectively, by the responsibility $Resp(e)$ in ascending order. Then, VEST prunes smallest $pr|\mathcal{G}|$ elements from core tensor and smallest $pr|\mathbf{A}^{(n)}|$ from each factor matrix, where pr is the pruning rate

of the current iteration. VEST starts with a very small pruning rate pr (INIT_PR) and slowly increases pr until maximum pruning rate (MAX_PR) is reached. The default values of INIT_PR and MAX_PR are set to 0.01 and 0.1, respectively. The slowly increasing sparsity strategy allows for elements with potential importance to the updated while iteratively pruning only the fraction of elements that we are certain are not important as the confidence of the certainty increases with the iteration (increasing pruning rate).

To determine when to stop pruning, VEST^{auto} (default VEST) determines the final sparsity automatically by tracking changes in the reconstruction error and determines to stop pruning when an elbow point of the reconstruction error curve is reached. The elbow point is estimated as the point when the second derivative of the RE curve, estimated as $(RE_t + RE_{t-2} - 2 * RE_{t-1})/pr_t$ where RE_t and pr_t are RE and pruning rate at t^{th} iteration, respectively, exceeds a small threshold (0.05 used). For testing, we also provide manual version of VEST, i.e., VEST^{man} takes a target sparsity s as an input from the user and stops pruning when the total sparsity reaches s. With the above conditions are satisfied, VEST iteratively updates factor matrices and core tensor without pruning until reconstruction error converges or the number of iterations exceed maximum iteration. The last update step without pruning can be considered as a fine tuning step.

D. Sparse Element-wise Update Rule

VEST updates elements of the core tensor and factor matrices based on a coordinate descent approach in parallel. It enables VEST to update the core tensor and factor matrices without changing the value of the pruned elements, i.e., guaranteeing non-decreasing sparsity. VEST checks the marking table which indicates whether elements have been pruned, and updates only the un-pruned elements. The update of an element is performed with observable tensor entries and fixed values of other elements in the factor matrices and the core tensor. The update rules for the core tensor and factor matrices are derived by setting the partial derivative of the loss function to zero and solving for each element. Advantages of our update rules are that 1) accuracy is high and convergence is faster,2) parallelization and selective updates are possible because all the elements are independently updated, and 3) the size of intermediate data is small, making the algorithm scalable.

For an element $a^{(n)}_{i_n j_n}$ of factor matrix $\mathbf{A}^{(n)}$, the element-wise update rule with L_1 regularization is provided in the following Lemmas.

Lemma 1 (Update rule for factor matrix with L_1 regularization)

$$\arg\min_{a^{(n)}_{i_n j_n}} L_1(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \begin{cases} (\lambda - g_{fm})/d_{fm} & \text{if } g_{fm} > \lambda \\ -(\lambda + g_{fm})/d_{fm} & \text{if } g_{fm} < -\lambda \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where

$$g_{fm} = 2\Big(\sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \mathcal{X}_\alpha \boldsymbol{\delta}^{(n)}_\alpha(j_n)\Big) - \Big(\sum_{\forall t \neq j_n} \boldsymbol{v}^{(n)}_{i_n j_n}(t) \cdot a^{(n)}_{i_n t}\Big), \tag{14}$$

$$d_{fm} = 2\boldsymbol{v}^{(n)}_{i_n j_n}(j_n), \tag{15}$$

$\boldsymbol{v}^{(n)}_{i_n j_n}$ is a length J_n vector whose jth element is

$$\boldsymbol{v}^{(n)}_{i_n j_n}(j) = \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \boldsymbol{\delta}^{(n)}_\alpha(j) \boldsymbol{\delta}^{(n)}_\alpha(j_n), \tag{16}$$

$\boldsymbol{\delta}^{(n)}_\alpha$ is a length J_n vector whose jth element is

$$\boldsymbol{\delta}^{(n)}_\alpha(j) = \sum_{\forall \beta_{j_n = j} \in \mathcal{G}} \mathcal{G}_{\beta_{j_n = j}} \prod_{k \neq n} a^{(k)}_{i_k j_k}, \tag{17}$$

$\Omega^{(n)}_{i_n}$ is the subset of Ω whose index of nth mode is i_n, and λ is a regularization parameter. □

For an element $\mathcal{G}_\beta$ of core tensor, the element-wise update rule with L_1 regularization is as follows:

Lemma 2 (Update rule for core tensor with L_1 regularization)

$$\arg\min_{\mathcal{G}_\beta} L_1(\mathcal{G}, \mathbf{A}^{(1)}, ..., \mathbf{A}^{(N)}) = \begin{cases} (\lambda - g_c)/d_c & \text{if } g > \lambda \\ -(\lambda + g_c)/d_c & \text{if } g < -\lambda \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

where $g_c = -2 \sum_{\forall \alpha \in \Omega} (\mathcal{X}_\alpha - \sum_{\forall \gamma \neq \beta} \mathcal{G}_\gamma \prod_{n=1}^{N} a^{(n)}_{i_n j_n}) \cdot \prod_{n=1}^{N} a^{(n)}_{i_n j_n}$, and $d_c = 2 \sum_{\forall \alpha \in \Omega} \big(\prod_{n=1}^{N} a^{(n)}_{i_n j_n}\big)^2$. □

The proofs of Lemmas 1 and 2 are provided in the Supplementary [25].

E. Parallel Update Algorithms

Responsibility calculation and factor matrices updates are performed in parallel. Algorithm 2 describes the pruning process where responsibility values of the core tensor and factor matrices are calculated in parallel for each observable entries of the input tensor. Note that the use of $B(\alpha)$ in line 5 enabled fast computing of $Resp(\mathcal{G}_\beta)$ in line 6; for a given β in line 4, computing line 5 requires $O(|\Omega|)$ rather than $O(|\Omega||\mathcal{G}|)$ since there is no need to compute $\sum_{\forall \beta \neq \gamma \in \mathcal{G}} \mathcal{G}_\gamma \prod_{n=1}^{N} a^{(n)}_{i_n j_n}$ in Eq. (7) from scratch.

The element-wise update of factor matrix $\mathbf{A}^{(n)}$ is performed in parallel for each rows of factor matrices using L_1 regularization (see Algorithm 3). Elements of a core tensor are highly dependent on each other and thus updating them cannot be made parallel, although a part of required computations can be made parallel (line 1 of Algorithm 4). However, considering that typical size $|\mathcal{G}|$ of the core tensor is small, the core tensor updates are a minor burden in the computational process. Element-wise update of the core tensor $\mathcal{G}$ using L_1 regularization is detailed in Algorithm 4.

Algorithm 2: Parallel Pruning

Input : Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$, core tensor $\mathcal{G} \in J_1 \times ... \times J_N$, and pruning rate pr.
Output: Pruned $\mathbf{A}^{(n)} (n = 1, \cdots, N)$ and $\mathcal{G}$

for $\alpha = \forall(i_1, ..., i_N) \in \Omega$ **do** ▷ **in parallel**
  calculate $B(\alpha) = \sum_{\forall \beta = (j_1, \cdots, j_N) \in \mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}$
  calculate $\mathcal{X}_\alpha - B(\alpha)$ ▷ Eq.(5)
for $\beta = \forall(j_1, ..., j_N) \in \mathcal{G}$ **do** ▷ **in parallel**
  calculate $\sum_{\forall \alpha \in \Omega} (\mathcal{X}_\alpha - B(\alpha) + \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n})$
  calculate $Resp(\mathcal{G}_\beta)$ ▷ Eq.(3), (7)
sort core tensor elements by $Resp(\mathcal{G}_\beta)$ values in an ascending order
for $i_n = 1...I_n$ **do**
  for $j_n = 1...J_n$ **do** ▷ **in parallel**
    for $\alpha = \forall(i_1, ..., i_N) \in \Omega^{(n)}_{i_n}$ **do**
      calculate $(2(\mathcal{X}_\alpha - B(\alpha)) + B_{j_n = j}(\alpha)) \cdot B_{j_n = j}(\alpha)$
    calculate $Resp(a^{(n)}_{i_n j_n})$ ▷ Eq.(3), (9)
  sort factor matrix elements by $Resp(a^{(n)}_{ij})$ values in an ascending order
prune smallest $pr|\mathcal{G}|$ and $pr|\mathbf{A}^{(n)}|$ elements of $\mathcal{G}$ and $\mathbf{A}^{(n)} (n = 1, ..., N)$, respectively.

Algorithm 3: Parallel element-wise factor matrix update

Input : Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$, and core tensor $\mathcal{G} \in J_1 \times ... J_N$.
Output: Updated factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$

for $n = 1...N$ **do** ▷ nth factor matrix
  for $i_n = 1...I_n$ **do** ▷ **in parallel**
    for $j_n = 1...J_n$ **do**
      if $a^{(n)}_{i_n j_n}$ *is pruned* **then**
        continue
      for $\alpha = \forall(i_1, ..., i_N) \in \Omega^{(n)}_{i_n}$ **do**
        for $\beta = \forall(j_1, ..., j_N) \in \mathcal{G}$ **do** ▷ compute $\boldsymbol{\delta}$
          $\boldsymbol{\delta}^{(n)}_\alpha(j_n) \longleftarrow \boldsymbol{\delta}^{(n)}_\alpha(j_n) + \mathcal{G}_\beta \prod_{\forall k \neq n} a^{(k)}_{i_k j_k}$
        accumulate $\mathcal{X}_\alpha \boldsymbol{\delta}^{(n)}_\alpha(j_n)$, and update $\boldsymbol{v}^{(n)}_{i_n j_n}$ ▷ Eq.(16), (17)
      update $a^{(n)}_{i_n j_n}$ using Eq. (13) for L_1

Algorithm 4: Parallel element-wise core tensor update

Input : Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} (n = 1, \cdots, N)$, and core tensor $\mathcal{G} \in J_1 \times ...J_N$.
Output: Updated core tensor $\mathcal{G} \in J_1 \times ...J_N$

for $\alpha = \forall(i_1, ..., i_N) \in \Omega$ **do** ▷ **in parallel**
  calculate $B(\alpha) = \sum_{\forall \beta = (j_1, \cdots, j_N) \in \mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}$
for $\beta = \forall(j_1, ..., j_N) \in \mathcal{G}$ **do**
  calculate $\sum_{\forall \alpha \in \Omega} (\mathcal{X}_\alpha - B(\alpha) + \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n}) \cdot \prod_{n=1}^{N} a^{(n)}_{i_n j_n}$
  calculate $\sum_{\forall \alpha \in \Omega} (\prod_{n=1}^{N} a^{(n)}_{i_n j_n})^2$
  update $\mathcal{G}_\beta$ using Eq. (19) for L_1

IV. Experiments

We conduct performance comparison to evaluate how accurately and sparsely VEST decompose a given tensor compared to other methods.

A. Experimental Settings

Datasets. We used three real-world datasets and synthetic datasets as summarized in Table II. The real-world datasets are MovieLens[1], Yelp[2], and AmazonFood[3]. MovieLens is a 4^{th} order tensor of movie ratings containing (user, movie, year, hour). Yelp is a 3^{rd} order tensor of business services rating data containing (user, business, year-month). AmazonFood is a 3^{rd} order tensor of food review scores from Amazon containing (product, user, year-month). As several sparse methods are not scalable and work only on three-order tensor, we used subsets of Yelp-s and AmazonFood-s to compare with all competitors and use the three full-scale data for comparing methods that are scalable. We also generated synthetic random tensors of various sizes and orders to test data scalability.

Competitors. We compared VEST with five published methods for sparse results as follows.

- PTucker-Approx [22] : scalable Tucker decomposition method for partially observable tensor.
- TTP [8]: A tensor decomposition method that results in sparse components based on value pruning.
- Sparse CP [3]: CP decomposition method with lasso penalty.
- ParCube [12] : scalable CP decomposition method to obtain sparse factor matrices.
- APG-NTD [26] : sparse non-negative tucker decomposition method that utilizes special Kronecker product structure.

We also compare VEST with the following four sparsification strategies which are methods implemented in the VEST framework and differs only in the sparsification strategy, i.e., they are implemented using the VEST missing value support and paralleling strategies.

[1]https://grouplens.org/datasets/movielens/
[2]http://www.yelp.com/dataset_challenge/
[3]http://snap.stanford.edu/data/web-FineFoods.html

Table II
SUMMARY OF DATASETS AND HYPERPARAMETERS USED.

Name	Order	Dimensionality	Ranks	$\|\Omega\|$	$\|\Omega\|_{test}$
MovieLens	4	$138K \times 27K \times 21 \times 24$	$6 \times 6 \times 2 \times 2$	18M	2M
Yelp	3	$71K \times 16K \times 108$	$10 \times 10 \times 10$	301K	33K
AmazonFood	3	$74K \times 256K \times 143$	$9 \times 9 \times 14$	511K	57K
Yelp-s	3	$50 \times 50 \times 10$	$5 \times 5 \times 5$	235	32
AmazonFood-s	3	$50 \times 50 \times 10$	$5 \times 5 \times 5$	444	51
Synthetic	$3-10$	$10^3 - 10^8$	$3 \times \cdots \times 3$	$10^3 - 10^7$	-

- Tucker-ALS [1]: Conventional Tucker factorization method (HOOI).
- L1 (Lasso): A Tucker factorization method with lasso sparsity constraint implemented as VEST^{man} with sparsity $s = 0$.
- Valuepruning$_{L1}$ and Valuepruning$_{LF}$: A Tucker factorization method with value pruning at the last step implemented as VEST^{man} with sparsity $s = 0$ followed by value pruning with ratio 0.6 for L_1 and L_F losses.

Environment. VEST was written in C++ with OPENMP [27] and ARMADILLO [28] libraries for parallelization. We initialize factor matrices and core tensor using $frand()$ function provided by C++. Methods L1 (Lasso) and Value Pruning were run on VEST framework with the difference just in the pruning approaches. We used the codes provided by the authors for TTP [8] (R) and Sparse CP [3] (Matlab). Tucker-ALS was performed via Tensor Toolbox for Matlab [29]. All experiments were done on a single machine equipped with an Intel Xeon E5-2630 v4 2.2GHz CPU (10 cores/20 threads) and 512GB memory. All reported measures are averages of five runs, unless otherwise stated.

B. Performance Comparison

We compared the accuracy and sparsity VEST^{auto} (default VEST) with VEST^{auto} (VEST with L_F regularization) and those of the competitors on 5 real world datasets: Yelp-s, AmazonFood-s, Yelp, AmazonFood, and MovieLens (Table II). Since TTP, Sparse CP, Tucker-ALS, and APG-NTD have scalability issues and are limited to order three tensors, those are compared with VEST on only Yelp-s and AmazonFood-s datasets.

We measured and compared normalized test cell reconstruction errors over observable entries in input tensors. VEST has a better performance than other methods in terms of sparsity and error as VEST is the closest method to the bottom-right region that indicates the best point. As shown in Fig. 1, VEST decomposed a given tensor with up to 13.4 times lower TestRE compared to other methods, excluding L1 (VEST w.o. pruning), at a similar sparsity with little difference between each other. Fig. 1 also shows that at a similar TestRE value, VEST outputs up to 2.6 times more sparse factor matrices and core tensor compared to other methods, where the sparsity is measured as the ratio of number of nonzero values in $\mathcal{G}$ and $\mathbf{A}^{(n)}$ over $|\mathcal{G}| + |\sum_{n=1}^{N} \mathbf{A}^{(n)}|$. Since VEST exploits the *responsibility* values to update and prune factor matrices and core tensor, VEST becomes a closer method to the best point than L1, Value Pruning, and PTucker-Approx. For all datasets, TTP, Sparse CP, Tucker-ALS, and APG-NTD achieve higher errors than VEST since they deal with the values of missing entries as zero and then compute tensor factorization for data including observable data and an enormous number of zeros. Moreover, they also have scalability issues so that they are compared with VEST on only Yelp-s and AmazonFood-s datasets. Although the factor matrices of ParCube are very sparse, ParCube which is sampling-based CP decomposition achieves a high error. ParCube focuses on analyzing sparse factor matrices, rather than predicting missing values.

V. CONCLUSION

We proposed VEST, a very-sparse Tucker factorization method for sparse and partially observable tensors. By deriving the element-wise partial differential equations, determining the importance of elements by responsibilities, and parallel distribution of computational work, VEST successfully offers very sparse and accurate results that are applicable for large partially observable tensors. VEST generates sparser and more accurate results for partially observable tensors compared to best performing sparse factorization competitors. Future works include better initialization for Tucker factorization, integration of prior knowledge, and effective visualization of tensor results.

ACKNOWLEDGMENT

Publication of this article has been funded by the National Research Foundation of Korea (2018R1A1A3A0407953, 2018R1A5A1060031) and by Korea Agency for Infrastructure Technology Advancement (KAIA) grant funded by the Ministry of Land, Infrastructure and Transport (19CTAP-C152020, 19CTAP-C152017).

REFERENCES

[1] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.
[2] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *ACM SIGKDD'16*, pp. 1135–1144, 2016.
[3] G. Allen, "Sparse higher-order principal components analysis," in *Artificial Intelligence and Statistics*, pp. 27–36, 2012.

[4] J. Lee, D. Choi, and L. Sael, "CTD: Fast, accurate, and interpretable method for static and dynamic tensor decompositions.," *PloS One*, vol. 13, no. 7, p. e0200579, 2018.

[5] J. Lee, S. Oh, and L. Sael, "GIFT: Guided and Interpretable Factorization for Tensors with an application to large-scale multi-platform cancer analysis," *Bioinformatics*, vol. 34, no. 24, pp. 4151–4158, 2018.

[6] O. H. Madrid-Padilla and J. Scott, "Tensor decomposition with generalized lasso penalties," *Journal of Computational and Graphical Statistics*, vol. 26, no. 3, pp. 537–546, 2017.

[7] M. Mørup, L. K. Hansen, and S. M. Arnfred, "Algorithms for sparse nonnegative tucker decompositions," *Neural computation*, vol. 20, no. 8, pp. 2112–2131, 2008.

[8] W. W. Sun, J. Lu, H. Liu, and G. Cheng, "Provable sparse tensor decomposition," *Journal of the Royal Statistical Society: Series B*, vol. 79, pp. 899–916, jun 2017.

[9] S. Yi, Z. Lai, Z. He, Y. ming Cheung, and Y. Liu, "Joint sparse principal component analysis," *Pattern Recognition*, vol. 61, no. 2, pp. 524–536, 2017.

[10] J. Gao, "Robust L1 principal component analysis and its bayesian variational inference," *Neural Computation*, vol. 20, no. 2, pp. 555–572, 2008.

[11] M. Zhang and C. H. Q. Ding, "Robust tucker tensor decomposition for effective image representation," in *ICCV*, pp. 2448–2455, 2013.

[12] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *ECML-PKDD*, vol. 7523 of *Lecture Notes in Computer Science*, pp. 521–536, Springer, 2012.

[13] M. W. Mahoney, M. Maggioni, and P. Drineas, "Tensor-CUR Decompositions for Tensor-Based Data," *SIAM J. Matrix Anal. Appl*, vol. 30, pp. 957–987, jan 2008.

[14] A. Pascual-Montano, J. M. Carazo, K. Kochi, D. Lehmann, and R. D. Pascual-Marqui, "Nonsmooth nonnegative matrix factorization (nsNMF)," *IEEE TPAMIT*, vol. 28, no. 3, pp. 403–415, 2006.

[15] Y.-D. Kim and S. Choi, "Nonnegative tucker decomposition," in *CVPR'07*, pp. 1–8, IEEE, 2007.

[16] N. Qi, Y. Shi, X. Sun, and B. Yin, "TenSR: Multi-dimensional Tensor Sparse Representation," *2016 IEEE CVPR*, pp. 5916–5925, 2016.

[17] Z. Zhang and S. Aeron, "Denoising and completion of 3d data via multidimensional dictionary learning," in *IJCAI*, pp. 2371–2377, IJCAI/AAAI Press, 2016.

[18] F. Jiang, X.-y. Liu, H. Lu, and R. Shen, "Efficient Multi-Dimensional Tensor Sparse Coding Using t-Linear Combination," in *AAAI 2018*, pp. 3326–3333, 2018.

[19] D. Lee, J. Lee, and H. Yu, "Fast tucker factorization for large-scale tensor completion," in *ICDM 2018*, pp. 1098–1103, 2018.

[20] J. Li, Y. Ma, X. Wu, A. Li, and K. J. Barker, "PASTA: A parallel sparse tensor algorithm benchmark suite," *CoRR*, vol. abs/1902.03317, 2019.

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

[22] S. Oh, N. Park, L. Sael, and U. Kang, "Scalable Tucker factorization for sparse tensors - algorithms and discoveries," in *ICDE*, (Paris, France), IEEE Computer Society, 2018.

[23] S. Smith and G. Karypis, "Accelerating the tucker decomposition with compressed sparse tensors," in *Euro-Par 2017*, vol. 10417 of *Lecture Notes in Computer Science*, pp. 653–668, Springer, 2017.

[24] T. G. Kolda, *Multilinear operators for higher-order decompositions*, vol. 2. United States. Department of Energy, 2006.

[25] "Vest - supplementary document." https://github.com/vestbigcomp/vestbigcomp2020/blob/master/supplementary.pdf.

[26] Y. Xu, "Alternating proximal gradient method for sparse nonnegative tucker decomposition," *Math. Program. Comput.*, vol. 7, no. 1, pp. 39–70, 2015.

[27] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, pp. 46–55, Jan. 1998.

[28] C. Sanderson and R. Curtin, "Armadillo: a template-based c++ library for linear algebra," *Journal of Open Source Software*, 2016.

[29] B. W. Bader, T. G. Kolda, and et la., "Tensor toolbox for matlab v. 3.0, version 00."