Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.031150

ARTICLE

Block Incremental Dense Tucker Decomposition with Application to Spatial and Temporal Analysis of Air Quality Data

SangSeok Lee[1], HaeWon Moon[1] and Lee Sael[1,2,*]

[1]Department of Artificial Intelligence, Ajou University, Suwon-si, 16499, Korea

[2]Department of Software and Computer Engineering, Ajou University, Suwon-si, 16499, Korea

*Corresponding Author: Lee Sael. Email: sael@ajou.ac.kr

Received: 17 May 2023 Accepted: 22 September 2023 Published: 30 December 2023

ABSTRACT

How can we efficiently store and mine dynamically generated dense tensors for modeling the behavior of multidimensional dynamic data? Much of the multidimensional dynamic data in the real world is generated in the form of time-growing tensors. For example, air quality tensor data consists of multiple sensory values gathered from wide locations for a long time. Such data, accumulated over time, is redundant and consumes a lot of memory in its raw form. We need a way to efficiently store dynamically generated tensor data that increase over time and to model their behavior on demand between arbitrary time blocks. To this end, we propose a Block Incremental Dense Tucker Decomposition (BID-Tucker) method for efficient storage and on-demand modeling of multidimensional spatiotemporal data. Assuming that tensors come in unit blocks where only the time domain changes, our proposed BID-Tucker first slices the blocks into matrices and decomposes them via singular value decomposition (SVD). The SVDs of the *time* × *space* sliced matrices are stored instead of the raw tensor blocks to save space. When modeling from data is required at particular time blocks, the SVDs of corresponding time blocks are retrieved and incremented to be used for Tucker decomposition. The factor matrices and core tensor of the decomposed results can then be used for further data analysis. We compared our proposed BID-Tucker with D-Tucker, which our method extends, and vanilla Tucker decomposition. We show that our BID-Tucker is faster than both D-Tucker and vanilla Tucker decomposition and uses less memory for storage with a comparable reconstruction error. We applied our proposed BID-Tucker to model the spatial and temporal trends of air quality data collected in South Korea from 2018 to 2022. We were able to model the spatial and temporal air quality trends. We were also able to verify unusual events, such as chronic ozone alerts and large fire events.

KEYWORDS

Dynamic decomposition; tucker tensor; tensor factorization; spatiotemporal data; tensor analysis; air quality

1 Introduction

Various real-world stream data come in the form of dense dynamic tensors, i.e., time-growing multi-dimensional arrays. Accumulated over time, these data require efficient storage and efficient on-demand processing. Tensor decomposition methods have been widely used to analyze and model from unlabeled multidimensional data. Several scalable methods have been proposed for efficient tensor

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

decomposition [1–4]. There are also several dynamic tensor decomposition methods. Most dynamic tensor decomposition methods focus on the CANDECOMP/PARAFAC (CP) decomposition method [5–8]. However, compared to Tucker decompositions, CP decompositions tend to produce higher reconstruction errors when the input tensors are skewed and dense [9]. Therefore, recent studies have focused on dynamic Tucker decompositions [10,11]. D-L1 Tucker [10] emphasizes outlier-resistant Tucker analysis of dynamic tensors, and D-TuckerO [11] focuses on the computational efficiency of dynamic Tucker decomposition. Overall, the need for efficient storage has been overlooked in previous dynamic methods, which focus on speed, accuracy, and scalability. Thus, there is still a need for scalable Tucker decomposition methods. These methods should allow efficient storage and partial on-demand modeling and analysis of dense and dynamically generated tensors.

Our proposed Block Incremental Dense Tucker Decomposition (BID-TUCKER) method efficiently stores and decomposes, on-demand, the multi-dimensional tensor data that are dynamically generated in the form of dense tensor blocks. Each incoming tensor block is transformed into slice matrices. Each of the slice matrices is then decomposed by a singular value decomposition (SVD) method. The approach of decomposing slice matrices was proposed in D-Tucker [12] for static tensors. We extend the approach to dynamically generated tensors. We store the block-wise SVD results of the slice matrices and use them, on-demand, to decompose the original tensor blocks (Fig. 1). Storing the SVD results instead of the original tensor blocks significantly reduces memory requirements. The Tucker decomposition result of the queried tensor blocks can be further analyzed and used for modeling. In this paper, we apply our method to spatiotemporal air quality tensor data and perform analysis to detect spatial and temporal trends.

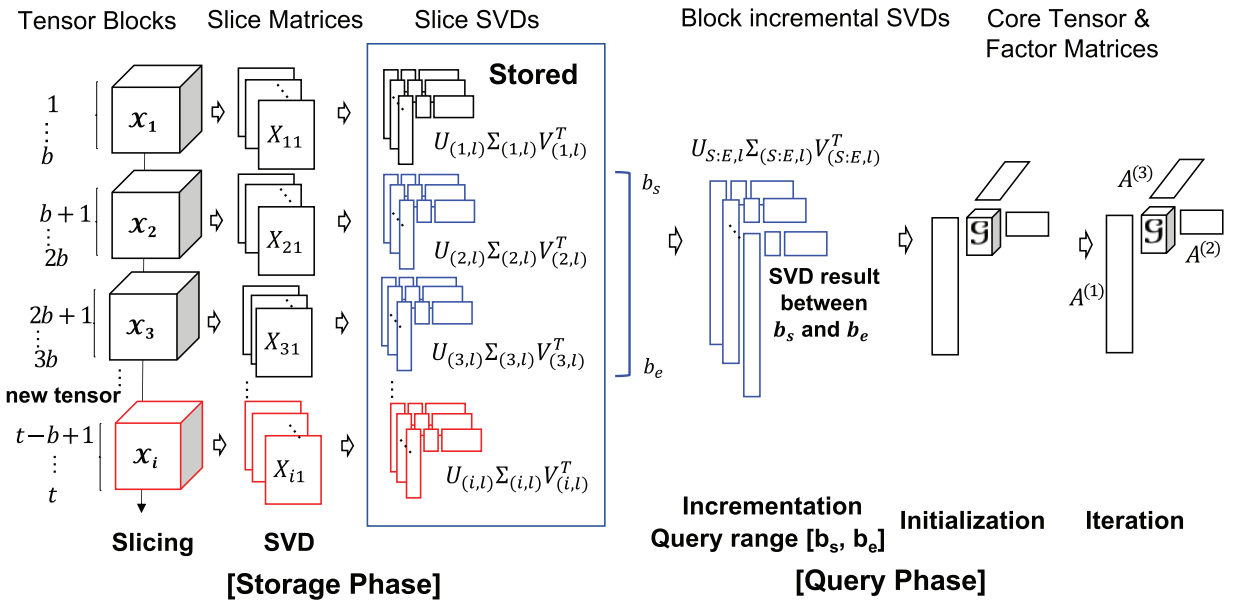

Figure 1: Storage phase and query phase of the proposed BID-TUCKER

Our contributions are summarized as follows:

- Propose an efficient storage method for block-wise generated tensor stream data.
- Propose an efficient on-demand dense tensor decomposition method for queried blocks for further analysis and modeling.

- Compare the reconstruction error and the computational speed of the proposed method with D-Tucker [12] and vanilla Tucker decomposition.
- Apply the proposed method to Korean air quality data and show how spatial and temporal trends can be modeled.

2 Background

We begin by summarizing the tensor operation and the terms used in the paper. The notations and descriptions are consistent with the notations of Kolda et al. [9] and also D-Tucker [12]. Table 1 lists the symbols used.

Table 1: Table of symbols consistent with the notations of Kolda et al. [9]

Symbol	Definition	Symbol	Definition
$\mathcal{X}$	Temporal tensor ($\in \mathbb{R}^{I_1 \times \cdots \times I_N}$)	$\mathcal{G}$	Core tensor ($\in \mathbb{R}^{J_1 \times \cdots \times J_N}$)
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 entry of $\mathbf{A}^{(n)}$
b	Temporal block size	$\mathbf{X}_{il}$	l-th slice matrix of ith time block
$\|\mathcal{X}\|_F$	Frobenius norm of tensor $\mathcal{X}$	$\|\mathcal{X}\|_1$	Sum of absolute values of tensor $\mathcal{X}$
$[\mathbf{X}, \boldsymbol{Y}]$	Vertical concatenation of matrices $\mathbf{X}$ and $\boldsymbol{Y}$	$[\mathbf{X}; \boldsymbol{Y}]$	Horizontal concatenation of matrices $\mathbf{X}$ and $\boldsymbol{Y}$
$\mathbf{U}_{(i,l)}$	Left singular value matrix of lth slice matrix in ith time block	$\mathbf{\Sigma}_{(i,l)}$	Singular value matrix of lth slice matrix in ith time block
$\mathbf{V}_{(i,l)}$	Right singular value matrix of lth slice matrix in ith time block	$\mathbf{U}_{(S:E,l)}$	Left singular value matrix of lth slice matrix in query time range
$\mathbf{\Sigma}_{(S:E,l)}$	Singular value matrix of lth slice matrix in query time range	$\mathbf{V}_{(S:E,l)}$	Right singular value matrix of lth slice matrix
S	Index of block tensor corresponding to query start-time block	E	Index of block tensor corresponding to query end-time block

2.1 Tensor Operation

Matricization transforms a tensor into a matrix. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted as $\mathbf{X}_{(n)}$. The mapping from an element $(i_1, ..., i_N)$ of $\mathcal{X}$ to an element (i_n, j) of $\mathbf{X}_{(n)}$ is given as follows:

$$j = 1 + \sum_{k=1, k \neq n}^{N} \left[(i_k - 1) \prod_{m=1, m \neq n}^{k-1} I_m \right]. \tag{1}$$

Given a tensor of the form $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \cdots \times I_N}$, **slice matrices** are two-dimensional slices of the tensor defined by fixing all but two indices. For example, in a 3^{rd}*order* tensor, the horizontal, lateral, and frontal slices of the tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ can be denoted as $\mathbf{X}_{i_1::}$, $\mathbf{X}_{:i_2:}$, and $\mathbf{X}_{::i_3}$, respectively. A mode-1 matrices $X_{(1)}$ of a tensor $\mathcal{X}$ can be represented as slice matrices as follows:

$$\mathbf{X}_{(1)} = [\mathbf{X}_{::1}; \cdots; \mathbf{X}_{::(I_{3 \times \cdots \times I_N})}] = [\mathbf{X}_1; \cdots \mathbf{X}_l; \cdots \mathbf{X}_L], \tag{2}$$

where $\mathbf{X}_l \in \mathbb{R}^{I_1 \times I_2}$, L is equal to $I_3 \times \cdots \times I_N$, and the index l is defined using the following equation [9]:

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

where K_m is the dimensionality of mode-m, N is the order of the input tensor, and $\prod_{m=3}^{i-1} K_m$ equals 1 if $i - 1 < m$. Note that on the right hand side of the Eq. (2) there is a slice matrix X_l denoted by the index l.

N-mode product allows multiplications between a tensor and a matrix. The n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J_n \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$. The resulting tensor has the shape of $I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N$. The element-wise n-mode product of a tensor can be written as follows:

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j_n i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} (\mathcal{X}_{i_1 i_2 \cdots i_N} \mathbf{U}_{j_n i_n}). \tag{4}$$

For more on tensor operations, please refer to a review by Kolda et al. [9].

2.2 *Block Incremental Singular Value Decomposition*

Incremental singular value decomposition (incremental SVD) is a method that generates the SVD result of a growing matrix in an incremental manner. Let $\mathbf{X}$ consist of two vertically concatenated blocks $\mathbf{X_A}$ and $\mathbf{X_B}$. Also, let $\mathbf{U_i}$, $\mathbf{\Sigma}_i$, and $\mathbf{V_i}^T$ be the SVD result of a given matrix i. We can compute the SVD of the vertically concatenated $\mathbf{X}$ as follows [13]:

$$\begin{aligned} \mathbf{X} = \begin{bmatrix} \mathbf{X_A} \\ \mathbf{X_B} \end{bmatrix} &\approx \begin{bmatrix} \mathbf{U_A \Sigma_A V_A}^T \\ \mathbf{U_B \Sigma_B V_B}^T \end{bmatrix} = \begin{bmatrix} \mathbf{U_A} & \mathbf{O} \\ \mathbf{O} & \mathbf{U_B} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma_A V_A}^T \\ \mathbf{\Sigma_B V_B}^T \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{U_A} & \mathbf{O} \\ \mathbf{O} & \mathbf{U_B} \end{bmatrix} \hat{U} \hat{\mathbf{\Sigma}} \hat{V}^T. \end{aligned} \tag{5}$$

With the above equation, the SVD result of $\mathbf{X}$ is as follows:

$$\mathbf{X} = \mathbf{U \Sigma V}^T \text{ where } \mathbf{U} = \begin{bmatrix} \mathbf{U_A} & \mathbf{O} \\ \mathbf{O} & \boldsymbol{U_B} \end{bmatrix} \hat{U}, \mathbf{\Sigma} = \hat{\mathbf{\Sigma}}, \text{ and } \mathbf{V}^T = \hat{V}^T. \tag{6}$$

2.3 *Tucker Decomposition*

Our proposed method is based on one of the most popular tensor decomposition methods, the Tucker decomposition [9]. Tucker decomposition decomposes tensors into factor matrices of each mode and a core tensor. Given an n^{th} order tensor $\mathcal{X}$ ($\in \mathbb{R}^{I_1 \times \cdots \times I_N}$), the Tucker decomposition approximates $\mathcal{X}$ to a core tensor $\mathcal{G}$ ($\in \mathbb{R}^{J_1 \times \cdots \times J_N}$) and factor matrices $\{\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n} | n = 1 \ldots N\}$. The core tensor $\mathcal{G}$ is much smaller than the input tensor $\mathcal{X}$ and the factor matrices $\mathbf{A}^{(n)}$ are normally column orthogonal. The Tucker decomposition with tensor operations is shown as follows:

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}. \tag{7}$$

2.4 *Higher-Order Orthogonal Iteration (HOOI) Algorithm*

A widely used technique for minimizing the reconstruction error in a standard Tucker decomposition is alternating least squares (ALS) [9], which updates a single factor matrix or core tensor while holding all others fixed. Algorithm 1 describes a vanilla Tucker factorization algorithm based

on ALS, called *higher-order orthogonal iteration* (HOOI) (see [9] for details). Algorithm 1 requires the storage of a full-density matrix $\mathcal{Y}_{(n)}$, and the amount of memory needed to store $\mathcal{Y}_{(n)}$ is $O(I_n \prod_{m \neq n} J_m)$. Also, computing a single n-mode product between an input tensor $\mathcal{X}$ and the factor matrices $A^{(n)}$, as in Eq. (8), requires $O(J_n \prod_{m \neq n} I_m)$ time complexity.

$$\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\boldsymbol{T}} \cdots \times_{n-1} \mathbf{A}^{(n-1)\boldsymbol{T}} \times_{n+1} \mathbf{A}^{(n+1)\boldsymbol{T}} \cdots \times_N \mathbf{A}^{(N)\boldsymbol{T}} \tag{8}$$

As the order, the mode dimensionality, or the rank of a tensor increases, the amount of memory required grows rapidly.

Algorithm 1: Tucker Decomposition via HOOI

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and core tensor dimensionality $J_1, ..., J_N$.
Output: Factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ $(n = 1, ..., N)$, and core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$.
:
1 initialize all factor matrices $\mathbf{A}^{(n)}$
2 **repeat**
3 **for** $n = 1...N$ **do**
4 $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\boldsymbol{T}} \cdots \times_{n-1} \mathbf{A}^{(n-1)\boldsymbol{T}} \times_{n+1} \mathbf{A}^{(n+1)\boldsymbol{T}} \cdots \times_N \mathbf{A}^{(N)\boldsymbol{T}}$
5 $\mathbf{A}^{(n)} \leftarrow J_n$ leading left singular values of $\mathcal{Y}_{(n)}$
6 **end**
7 **until** *reconstruction error converges or exceeds maximum iteration*;
8 $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\mathrm{T}} \cdots \times_N \mathbf{A}^{(N)\mathrm{T}}$

2.5 Sequentially Truncated HOSVD Algorithm

Sequentially Truncated Higher Order SVD (ST-HOSVD) [14] calculates the HOSVD in sequentially truncated form as follows in the Algorithm (2) using randomized SVD.

Algorithm 2: ST-HOSVD Algorithm [14]

Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and core tensor dimensionality $J_1, ..., J_N$.
Ouput: Factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ $(n = 1, ..., N)$, and core tensor
:
1 $\mathcal{G} \leftarrow \mathcal{X}$
2 **for** $n = 1...N$ **do**
3 $\mathcal{G}_{(n)} \approx [\mathbf{U}_1 \mathbf{U}_2] \begin{bmatrix} \mathbf{\Sigma}_1 & \\ & \mathbf{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{bmatrix}$, with $\mathbf{U}_1 \in \mathbb{R}^{I_n \times J_n}$
4 $\mathbf{A}^{(n)} \leftarrow \mathbf{U}_1$ // J_n `leading left singular values of` $\mathcal{G}_{(n)}$
5 $\mathcal{G}_{(n)} \leftarrow \mathbf{\Sigma}_1 \mathbf{V}_1^T$
6 **end**

The ST-HOSVD algorithm provides a way to sequentially obtain factor matrices without the repeated mode-n products between the original tensor and all other factor matrices, as in line 4 of the Algorithm 1.

3 Block Incremental Dense Tucker Decomposition

Our proposed Block Incremental Dense Tucker Decomposition (BID-Tucker) method assumes that only the time mode increases in dimension and dimensions remain for the rest of the modes. Our

BID-Tucker is divided into two phases (Fig. 1 and Algorithm 3): the storage phase and the query phase. In the storage phase, incoming tensor blocks are stored as slice SVDs. The slice SVDs are then used in the query phase to efficiently decompose tensor blocks. In the query phase, a dense Tucker decomposition is performed on the user-specified query blocks. The decomposition results can be used for further analysis such as anomaly detection. We explain our approach for both phases in detail.

Algorithm 3: Proposed BID-Tucker adopted from D-Tucker [12]

Input: Temporal tensor blocks of length b for storage phase or slice-SVDs covering query block indices $[S,E]$ for query phase

Output: Slice-SVDs for storage phase or factor matrices $\mathbf{A}^{(i)}(i = 1, 2, ..., N)$ and core tensor $\mathcal{G}$ of $[S,E]$ blocks for query phase

Parameter a SVD truncation size r, tensor ranks J_i $(i = 1...N)$, and a error tolerance ϵ

:

Storage Phase (SP):

store incoming tensor blocks in the form of slice-SVDs by Algorithm 4

Query Phase (QP):

block increment SVDs for query blocks [S,E] and initialize factor matrices $A^{(i)}(i = 1, 2, ..., N)$ by Algorithm 5

repeat

update factor matrices $\mathbf{A}^{(i)}(i = 1, 2, ..., N)$ and $\mathcal{Y}_{reuse}$ with Algorithm 6

until *the maximum iteration is reached or the error difference is smaller than the error tolerance* ε;

$\mathcal{G} \leftarrow \mathcal{Y}_{reuse} \times_3 \mathbf{A}^{(3)T} ... \times_N A^{(N)T}$

3.1 Storage Phase

Our storage phase, inspired by the block incremental SVD in FAST [13] and the approximation step of D-Tucker [12], provides a method to efficiently store the incoming tensor blocks. When a new tensor block arrives at the input system, the tensor is first sliced, where the two unfixed indices are time and space. Next, each of the sliced matrices is decomposed by the SVD. Only the SVD results are stored: singular values containing rectangular matrices $\mathbf{\Sigma}$s, left singular vector matrices $\mathbf{U}$s, and right singular vector matrices $\mathbf{V}$s.

Assuming there are B blocks and the SVD truncation size is R, the space requirement is $O(BI_1RL)$, where I_1 is the time mode dimension and $L = I_3 \times \cdots \times I_N$ is the number of sliced matrices for the input tensor $\mathcal{X}_b \in^{I_1 \times \cdots \times I_N}$. If $O(BI_1RL)$ is small enough, they can be stored in memory. Otherwise, the SVD results can be stored on disk for later retrieval. For our experiments, we assume that the SVD results stored in memory are used in the query phase for Tucker decomposition and analysis.

The storage phase is described in detail in the Algorithm 4. In the storage phase, given the spatiotemporal tensor data, the first mode is selected to be time, and the second mode is selected to be space (line 1). Once the two modes are selected, they are not changed during the storage and query phases. Afterward, the data is normalized so that all mode values are scaled, and a tensor is constructed (line 2). For each of the incoming tensor blocks, slice matrices are formed using the two modes, while the indices for the remaining modes are fixed (line 3). Each of the *slice matrices* is further decomposed with SVD, denoted as *slice-SVDs* (lines 4–6). Several SVD methods can be used. For smaller and less noisy tensor blocks, exact SVD can be used. For larger, noisy data, faster low-rank SVD methods such as randomized SVDs are recommended. Randomized SVDs are fast and memory

efficient, and their low-rank approximation helps remove noise. Since most of the real-world sensor data is large and noisy, we used the randomized SVDs to generate the slice SVDs. We extend this process to each new incoming block of tensors. The process is shown in the storage phase of Fig. 1.

Algorithm 4: Storage phase algorithm

Input: kth spatio-temporal block tensor of block length b
Output: Slice-SVDs $\mathbf{U}_{(k,l)}$, $\mathbf{\Sigma}_{(k,l)}$, and $\mathbf{V}_{(k,l)}$ of kth block for $l = 1 \ldots L$
Parameter SVD rank r
:
set mode-1 to time and mode-2 to space
construct normalized tensor block $\mathcal{X}_k$ from multi-dimensional spatio-temporal data
construct slice matrices $\mathbf{X}_{kl} \in \mathbb{R}^{I_1 \times I_2}$ following Eq. (2) and (3)
for $l \leftarrow 1$ **to** L **do**
    $\mathbf{U}_{(k,l)}, \mathbf{\Sigma}_{(k,l)}, \mathbf{V}_{(k,l)} \leftarrow SVD(\mathbf{X}_{kl})$
end
return $\mathbf{U}_{(k,l)}$, $\mathbf{\Sigma}_{(k,l)}$, and $\mathbf{V}_{(k,l)}$ *for* $l = 1 \ldots L$

3.2 Query Phase

In the query phase, core tensor and factor matrices of the queried block tensors are generated on-demand. The stored slice SVDs of blocks in the query time range are retrieved and processed in two steps: initialization and iteration. First, BID-Tucker constructs a single slice-SVDs by block-wise incrementing the slice-SVDs of the requested time blocks [S, E], which are used to initialize the factor matrices. Our BID-Tucker then iterates over the factor matrices to find optimal cell values. The core tensor is computed after the last iteration.

3.2.1 Initialization

The left figure in the query phase of Fig. 1 shows the block-wise incremented SVDs and the initialization of our BID-Tucker. In the block-wise increment step, the block-wise slice-SVDs are used to generate slice-SVDs that cover the entire query blocks [S, E].

Algorithm 5: Initialization algorithm of BID-Tucker adopted from FAST [13] and D-Tucker [12]

Input: Start time block index S, end time block index S, block size b, and SVD results $\mathbf{U}_{(k,l)}$, $\boldsymbol{S}_{(k,l)}$, and $\mathbf{V}_{(k,l)}$ of all slices $l = 1, \ldots, L$ and time blocks k between *[S,E]*.
Output: Initialized factor matrices $\mathbf{A}^{(i)}(i = 1, 2, ..., N)$ of the query blocks *[S,E]*
Parameter: SVD truncation size r, tensor ranks $J_i(i = 1, 2, ..., N)$
:
`/* construct block-wise incremented SVDs */`
for $l \leftarrow 1$ **to** L **do**
    $\mathbf{U}_{(r,l)}, \mathbf{\Sigma}_{(r,l)}, \mathbf{V}^T_{(r,l)} \leftarrow SVD([\mathbf{\Sigma}_{(S,l)}\mathbf{V}^T_{(S,l)}, \mathbf{\Sigma}_{(S+1,l)}\mathbf{V}^T_{(S+1,l)}, \ldots, \mathbf{\Sigma}_{(E,l)}\mathbf{V}^T_{(E,l)}])$
    $\mathbf{V}_{(S:E,l)} \leftarrow \mathbf{V}_{(r,l)}, \quad \mathbf{\Sigma}_{(S:E,l)} \leftarrow \mathbf{\Sigma}_{(r,l)}$
    block-wise split $\mathbf{U}_{(r,l)} = [\mathbf{U}_{(r(S),l)}, \mathbf{U}_{(r(S+1),l)}, \ldots, \mathbf{U}_{(r(E),l)}]$
    $\mathbf{U}_{(S:E,l)} \leftarrow [\mathbf{U}_{(S,l)}\mathbf{U}_{(r(S),l)}, \mathbf{U}_{(S+1,l)}\mathbf{U}_{(r(S+1),l)}, \ldots, \mathbf{U}_{(E,l)}\mathbf{U}_{(r(E),l)}]$
end
`/* initialize factor matrices */`
$\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \leftarrow \text{SVD}([\mathbf{U}_{(S:E,1)}\mathbf{\Sigma}_{(S:E,1)}; \ldots; \mathbf{U}_{(S:E,L)}\mathbf{\Sigma}_{(S:E,L)}])$

(Continued)

Algorithm 5 (continued)

8 $\mathbf{A}^{(1)} \leftarrow \mathbf{U}$
9 $\boldsymbol{Y}_{prev} = \mathbf{A}^{(1)T}[\mathbf{U}_{(S:E,1)}; \ldots; \mathbf{U}_{(S:E,L)}]$
10 $\boldsymbol{Y}_{(2),inter} \leftarrow \boldsymbol{Y}_{prev} blkdiag(\{\mathbf{\Sigma}_{(S:E,L)}\mathbf{V}^T_{(S:E,L)}\}^L_{l=1})$
11 $\mathcal{Y} \leftarrow reshape(\boldsymbol{Y}_{(2),inter}, [J_1, I_2, I_3, \ldots, I_N])$
12 $\mathbf{A}^{(2)} \leftarrow J_2$ leading singular vectors of $\boldsymbol{Y}_{(2)}$ from $\mathcal{Y}$
13 **for** $i \leftarrow 3$ **to** N **do**
14 **if** $i = 3$ **then**
15 $\boldsymbol{Y}_{prev} \leftarrow \boldsymbol{Y}_{prev} blkdiag(\{\mathbf{\Sigma}_{(S:E,l)}\mathbf{V}^T_{(S:E,l)}\mathbf{A}^{(2)}\}^L_{l=1})$
16 $\mathcal{Y} \leftarrow reshape(\boldsymbol{Y}_{prev}, [J_1, J_2, I_3, \ldots, I_N])$
17 **else**
18 $\mathcal{Y} \leftarrow \mathcal{Y}_{prev} \times_{i-1} \mathbf{A}^{(i-1)T}$
19 $\mathbf{A}^{(i)} \leftarrow J_i$ leading singular vectors of $\boldsymbol{Y}_{(i)}$ from $\mathcal{Y}$
20 $\mathcal{Y}_{prev} \leftarrow \mathcal{Y}$
21 **end**
22 **return** $\mathbf{A}^{(i)}(i = 1 \ldots N)$

The specifics of block-wise incremented SVD are described in the following. Let $[U_{(S,l)}; \ldots; U_{(E,l)}]$; be the left singular vectors of the l-th slice matrix between the start block S and the end block E. All left singular vectors in the requested time blocks are concatenated into the block diagonal form as in Eq. (9).

$$blkdiag(\{\mathbf{U}_l\}^E_{l=S}) = \begin{bmatrix} \mathbf{U}_{(S,l)} & 0 & \ldots & 0 \\ 0 & \mathbf{U}_{(S+1,l)} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & 0 & \mathbf{U}_{(E,l)} \end{bmatrix} \tag{9}$$

Then the l-th slice matrix corresponding to the time query blocks $[S, E]$ is derived in the following:

$$X_{(S:E,i)} \approx \begin{bmatrix} \mathbf{U}_{(S,l)}\mathbf{\Sigma}_{(S,l)}\mathbf{V}^T_{(S,l)} \\ \mathbf{U}_{(S+1,l)}\mathbf{\Sigma}_{(S+1,l)}\mathbf{V}^T_{(S+1,l)} \\ \vdots \\ U'_{(E,il)}\mathbf{\Sigma}_{(E,l)}\mathbf{V}^T_{(E,l)} \end{bmatrix} = blkdiag(\{\mathbf{U}_{(b,l)}\}^E_{b=S}) \begin{bmatrix} \mathbf{\Sigma}_{(S,l)}\mathbf{V}^T_{(S,l)} \\ \mathbf{\Sigma}_{(S+1,l)}\mathbf{V}^T_{(S+1,l)} \\ \vdots \\ \mathbf{\Sigma}_{(E,l)}\mathbf{V}^T_{(E,l)} \end{bmatrix}$$
$$= blkdiag\left(\{\mathbf{U}_{(i,l)}\}^E_{i=S}\right)\mathbf{U}_{(r,l)}\mathbf{\Sigma}_{(r,l)}\mathbf{V}^T_{(r,l)} = \mathbf{U}_{(S:E,l)}\mathbf{\Sigma}_{(S:E,l)}\mathbf{V}^T_{(S:E,l)}. \tag{10}$$

The $[\mathbf{\Sigma}'_{(S,l)}\mathbf{V}'^T_{(S,l)}; \mathbf{\Sigma}_{(S+1,l)}\mathbf{V}^T_{(S+1,l)}; \ldots; \mathbf{\Sigma}'_{(E,l)}\mathbf{V}'^T_{(E,l)}]$ is decomposed and expressed as $\mathbf{U}_{(r,l)}$, $\mathbf{\Sigma}_{(r,l)}$, $\mathbf{V}^T_{(r,l)}$ (line 2). Singular value matrices and right singular vector matrices are assigned to be the newly decomposed values as in $\mathbf{\Sigma}_{(S:E,l)} = \mathbf{\Sigma}_{(r,l)}$ and $V^T_{(S:E,l)} = \mathbf{V}^T_{(r,l)}$ (line 3). To update the left singular vector matrices, existing block diagonal matrices $blkdiag(\{\mathbf{U}_{(i,l)}\}^E_{i=S})$ are multiplied by block-wise split $\mathbf{U}_{(r,l)}$ matrices (line 4 and 5).

The next step in the initialization process is to initialize the cell values for the factor matrices and the core tensor of the query time range for efficient computation. There are two factors to consider when initializing. The first factor is that we do not want to refer to the original tensor values since we discard them after the storage phase. The second factor is that we want to initialize well so that the number of iterations required in the alternating least squares (ALS) procedure is minimal. D-Tucker [12] provides a way to decompose the tensor blocks with slice SVDs, which addresses both purposes. The method showed tolerable reconstruction error and a reduced number of iterations in ALS [12]. In

fact, our factor matrix initialization and iteration steps follow the initialization and iteration phases of the D-Tucker [12], modified higher-order orthogonal iteration (HOOI) algorithm (Algorithm 1). We rewrite them here for clarity of the proposed BID-Tucker.

The computational bottleneck of naive HOOI is the computation of a single n-mode product between the input tensor $\mathcal{X}$ and the factor matrices (line 4 of Algorithm 1). Again, there are two problems with the naive HOOI approach: it requires the input tensor $\mathcal{X}$ and the space complexity is high. Fortunately, the input tensor $\mathcal{X}$ can be approximated with the stored slice SVDs for each block without reconstruction, which also improves the time and space complexity. This is possible by approximating the mode-n tensor with slice matrices and replacing the regular higher-order singular value decomposition (HOSVD) approach used in regular HOOI (lines 5 and 8 of Algorithm 1) with the sequentially truncated higher-order singular value decomposition (ST-HOSVD) [14] approach. ST-HOSVD sequentially assigns the truncated left singular vector matrix of the mode-n tensor as the mode-n factor matrix. first mode More specifically, the first mode factor matrix $A^{(1)}$ can be obtained by first representing the mode-1 tensor $X_{(1)}$ as follows:

$$\begin{aligned}\mathcal{X} = \mathbf{X}_{(1)} &= [\mathbf{X}_1; \ldots; \mathbf{X}_l; \ldots; \mathbf{X}_L] \\ &\approx [\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T; \ldots; \mathbf{U}_l\mathbf{\Sigma}_l\mathbf{V}_l^T; \ldots; \mathbf{U}_L\mathbf{\Sigma}_L\mathbf{V}_L^T] \\ &= [\mathbf{U}_1\mathbf{\Sigma}_1; \ldots; \mathbf{U}_l\mathbf{\Sigma}_l; \ldots; \mathbf{U}_L\mathbf{\Sigma}_L] \times (blkdiag(\{\mathbf{V}_l^T\}_{l=1}^L)) \\ &\approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T(blkdiag(\{\mathbf{V}_l^T\}_{l=1}^L)) \end{aligned} \tag{11}$$

where L is equal to $I_1 \times .. \times I_N$, l is as defined in Eq. (3), $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the SVD result of $[\mathbf{U}_1\mathbf{\Sigma}_1; \ldots; \mathbf{U}_l\mathbf{\Sigma}_l; \ldots; \mathbf{U}_L\mathbf{\Sigma}_L]$, and $blkdiag(\{\mathbf{V}_l^T\}_{l=1}^L)$ is the block diagonal form of right singular vectors V_ls similar to Eq. (9). With the above mode-1 tensor $\mathbf{X}_{(1)}$ representation and application of ST-HOSVD, we can obtain the initial factor matrix for the first mode as $\mathbf{A}^{(1)} = U$ (line 8).

The initialization of the mode-2 factor matrix $\mathbf{A}^{(2)}$ also follows the ST-HOSVD and respects the order of the tensor operations. As with the first mode factor matrix, $\mathbf{A}^{(2)}$ is initialized to be the left singular vectors of the mode-2 matrices of $\mathcal{X} \times_1 A^{(1)}$, but without explicit reconstruction of $\mathcal{X}$ as in Eq. (12).

$$\begin{aligned}\mathcal{X} \times_1 \boldsymbol{A}^{(1)T} = \mathbf{A}^{(1)T}\mathbf{X}_{(1)} &\approx \mathbf{A}^{(1)T}[\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T; \ldots; \mathbf{U}_l\mathbf{\Sigma}_l\mathbf{V}_l^T; \ldots; \mathbf{U}_L\mathbf{\Sigma}_L\mathbf{V}_L^T] \\ &= \left(\mathbf{A}^{(1)T}[\mathbf{U}_1; \ldots; \mathbf{U}_l; \ldots; \mathbf{U}_L]\right) blkdiag(\{\mathbf{\Sigma}_l\boldsymbol{V}^{\boldsymbol{T}}{}_l\}_{l=1}^L) \\ &= Y_{(2),inter}blkdiag(\{\mathbf{\Sigma}_l\mathbf{V}_l^T\}_{l=1}^L) \end{aligned} \tag{12}$$

where $Y_{(2),inter} = \mathbf{A}^{(1)T}[\mathbf{U}_1; \ldots; \mathbf{U}_l; \ldots; \mathbf{U}_L]$, L is equal to $I_3 \times \cdots \times I_N$, $blkdiag(\{\mathbf{\Sigma}_l\mathbf{V}_l\}_{l=1}^L)$ is a block diagonal matrix. Transforming the right-most equation in Eq. (12) to $\mathbb{R}^{J_1 \times I_2 \times \cdots \times I_N}$ and taking the left singular vectors from mode-2 matricized from gives the initial values for $\mathbf{A}^{(2)}$ (lines 9–11).

Initialization of the mode-3 factor matrix $\mathbf{A}^{(3)}$, following the ST-HOSVD, is also similar. As with the mode-1 and mode-2 factor matrice, $\mathbf{A}^{(3)}$ is initialized to be the left singular vectors of mode-3 matricization of $\mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)}$ again without explicit reconstruction of $\mathcal{X}$ as in Eq. (13).

$$\begin{aligned}\mathcal{X} \times_1 \boldsymbol{A}^{(1)T} \times_2 \boldsymbol{A}^{(2)T} &= \mathbf{A}^{(1)T}\mathbf{X}_{(1)} blkdiag(\{\mathbf{A}^{(2)}\}_{l=1}^L) \\ &\approx \left(\mathbf{A}^{(1)T}[\mathbf{U}_1; \ldots; \mathbf{U}_L]\right) blkdiag(\{\mathbf{\Sigma}_l\mathbf{V}_l^T\mathbf{A}^{(2)}\}_{l=1}^L) \\ &= Y_{(2),inter}blkdiag(\{\mathbf{\Sigma}_l\mathbf{V}_l^T\mathbf{A}^{(2)}\}_{l=1}^L) \end{aligned} \tag{13}$$

where $Y_{(2),inter} = \mathbf{A}^{(1)T}[\mathbf{U}_1; \ldots; \mathbf{U}_l; \ldots; \mathbf{U}_L]$, L is equal to $I_3 \times \cdots \times I_N$, $blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l\}_{l=1}^{L})$ is a block diagonal matrix. Reshaping the rightmost equation in Eq. (13) to $\mathbb{R}^{J_1 \times J_2 \times \cdots \times I_N}$ and taking left-singular vectors from mode-3 matricized from gets the initial values for $\mathbf{A}^{(3)}$ (lines 15, 16, 19).

The remaining modes $i = 4, \ldots, N$ following the ST-HOSVD can also be initialized by taking the left singular vectors of mode-i in the matricized form of $\mathcal{X} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \times_{i-1} \mathbf{A}^{(i-1)T}$ (line 13). For efficient computation, D-Tucker [12] avoids redundant computation by finding the recursive computation pattern of $\mathcal{Y}_{prev} \times_{i-1} \mathbf{A}^{(i-1)T}$. That is, the recursive computation of $\mathcal{Y}_{prev}$ to mode-3 (line 15) is just the mode-$i-1$ product between $\mathcal{Y}_{prev}$ and $\mathbf{A}^{(i-1)T}$ (line 18).

3.2.2 Iteration

After initialization, to reduce reconstruction error, ALS is performed over the factor matrices to generate optimal cell values for factor matrices $\mathbf{A}^{(i)}(i = 1...N)$. We follow the iteration procedure of the D-Tucker [12], which follows the HOOI iteration process that (1) approximates the original tensor $\mathcal{X}$ with stored SVD results (2) carefully arranges the update for the first- and the second-factor matrices by utilizing SVD results and their transpose (3) and avoid duplicate calculations of the intermediate tensor.

The overall ALS iteration process is shown in Algorithm 6, which we describe in detail. Again, the algorithm and description follow those of D-Tucker [12] which is an approximate form of the HOOI Algorithm 1. HOOI constructs an intermediate tensor that $\mathcal{Y}$ by n-mode products of the original tensor and the factor matrices, except for the factor matrix that is to be updated. D-Tucker approximates this process for the updated first-factor matrix by computing the approximate of $\mathcal{X} \times_2 \mathbf{A}^{(2)T}$ with the previous factor matrix $\mathbf{A}^{(2)T}$ and transposing the SVD of stitched matrices (lines 3–5). Then, the rest of the factor matrices are multiplied normally (line 10), prior to the factor matrix update (line 11). The second-factor matrix is updated similarly (lines 7–11). Prior to the update of the rest of the factor matrices, intermediate tensor $\mathcal{Y}_{prev}$ that approximates $\mathcal{X} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T}$ is calculated for repeated reuse (lines 13, 14). Then the algorithm follows the same procedure as the HOOI (lines 16–17). Standard SVD is used for finding the leading singular vector of $\boldsymbol{Y}_{(i)}$ (lines 11, 17).

Algorithm 6: Iteration algorithm BID-Tucker rewritten from iteration phase of D-Tucker [12]

Input: factor matrices $\mathbf{A}^{(i)}(i = 1, ..., N)$ of the query blocks [S,E] from previous step
Output: updated $\mathbf{A}^{(i)}(i = 1, ..., N)$ and $\mathcal{Y}_{prev}$

1 **for** $i \leftarrow 1$ to 2 **do**
2 **if** $i = 1$ **then**
3 $\boldsymbol{Y}_{prev} = \mathbf{A}^{(2)T}[\mathbf{V}_1; \ldots; \mathbf{V}_L]$
4 $\boldsymbol{Y}_{(1),iter} \leftarrow \boldsymbol{Y}_{prev} blkdiag(\{\mathbf{\Sigma}_l \mathbf{U}_l^T\}_{l=1}^{L})$
5 $\mathcal{Y} \leftarrow reshape(\boldsymbol{Y}_{(1),iter}, [I_1, J_2, I_3, \ldots, I_N])$
6 **else**
7 $\boldsymbol{Y}_{prev} = \mathbf{A}^{(1)T}[\mathbf{U}_1; \ldots; \mathbf{U}_L]$
8 $\boldsymbol{Y}_{(2),iter} \leftarrow \boldsymbol{Y}_{prev} blkdiag(\{\mathbf{\Sigma}_l \mathbf{V}_l^T\}_{l=1}^{L})$
9 $\mathcal{Y} \leftarrow reshape(\boldsymbol{Y}_{(2),iter}, [J_1, I_2, I_3, \ldots, I_N])$
10 $\mathcal{Y} \leftarrow \mathcal{Y} \times_3 \mathbf{A}^{(3)T} \cdots \times_N \mathbf{A}^{(N)T}$
11 $\mathbf{A}^{(i)} \leftarrow J_i$ leading singular vectors of $\boldsymbol{Y}_{(i)}$ from $\mathcal{Y}$
12 **end**

(Continued)

Algorithm 6 (continued)

$\boldsymbol{Y}_{prev} \leftarrow \boldsymbol{Y}_{prev} blkdiag(\{\boldsymbol{\Sigma}_l \mathbf{V}_l^T \mathbf{A}^{(2)}\}_{l=1}^{L})$
$\mathcal{Y}_{prev} \leftarrow reshape(\boldsymbol{Y}_{prev}, [J_1, J_2, I_3, \ldots, I_N])$
for $i \leftarrow 3$ **to** N **do**
    $\mathcal{Y} \leftarrow \mathcal{Y}_{prev} \times_3 \mathbf{A}^{(3)T} \cdots \times_{i-1} \mathbf{A}^{(i-1)T} \times_{i+1} \mathbf{A}^{(i+1)T} \cdots \times_N \mathbf{A}^{(N)T}$
    $\mathbf{A}^{(i)} \leftarrow J_i$ leading singular vectors of $\boldsymbol{Y}_{(i)}$ from $\mathcal{Y}$
end
return $\mathbf{A}^{(i)}(i = 1...N)$ \text{and} $\mathcal{Y}_{prev}$

3.3 Complexity Analysis of BID-Tucker

To simplify the theoretical analysis of our BID-Tucker, we assume that the ranks of the output tensor are all J dimensional and that the dimensionality of a block of the input tensor is as follows: the dimension of the time mode is I and the dimension of remaining modes are all K, where $I \leq K \leq J > 0$.

The **storage phase** takes $O(BI^2K^{N-2})$ time and $O(BIRK^{N-2})$ space to process and store B blocks of the N^th order input tensor $\mathcal{X}_b \in^{I \times K \times \cdots \times K}$. Because the randomized SVD takes $O(I_1^2)$ [15] time, and the randomized SVD is performed on each of the $L = K^{N-2}$ slice matrices for a block of the input tensor. The space requirement for the storage phase, given the SVD rank R, is $O((IR + RR + KR)K^{N-2})$, since IR, RR, KR stand for the sizes of the left singular vectors, the singular values, and the right singular vectors of each of the K^{N-2} sliced matrices. The space complexity can be simplified to $O(IRK^{N-2})$.

The **query phase** takes $O(MNZIK^{N-2}J^2)$ time and $O(ZIK^{N-2}J)$ space, where I is the dimensionality of the input tensor, J is the output rank, Z is the number of tensor time blocks processed, M is the number of iterations, and N is the order of the input tensor. The query phase analysis of BID-Tucker largely follows the complexity analysis of D-Tucker, which takes $O(IK^{N-2}J^2)$ time for initialization, $O(MNIK^{N-2}J^2)$ time for iteration, and $O(IK^{N-2}J)$ space for processing a single block, $B = Z = 1$, of the input tensor [12].

4 Anomaly Detection via Tensor Decomposition

After obtaining the core tensors and factor matrices of the desired time range $[S, E]$, they can be used for spatial and temporal analysis. For the temporal analysis, we focus on finding unusual events. To do this, we define the anomaly score and the anomaly threshold based on the clusters of the time mode factor matrix. In a time mode factor matrix, a row contains latent values for the corresponding time point. Finding time-wise anomalies means finding unusual air quality that covers wide locations and features that differ from normal times.

The **anomaly score**: for each row of the time mode factor matrix is calculated by computing the minimum distance-to-centroid after clustering the row vectors. An appropriate clustering method can be selected after examining the distribution of the row vectors. We used k-means clustering but other clustering methods can be used. Then, for each cluster, a centroid vector $\mathbf{c}_i \in \mathbf{C}$ is constructed for $i = 1 \ldots K$, where K is the number of clusters. Given a row vector $\mathbf{v}$, the anomaly score is then defined as the distance of the vector to the nearest cluster. Specifically, we used the following score based on Euclidean distance:

$$as(\mathbf{v}) = \min_{\mathbf{c}_i \in \mathbf{C}} ||\mathbf{v} - \mathbf{c}_i||_2. \tag{14}$$

Different distance measurements can be used.

The **anomaly threshold** can be selected, after calculating the anomaly scores, to detect unusual events. Since the distribution of distances follows the Gaussian distribution, we use unbiased estimates of the Gaussian distribution, i.e., the mean $\boldsymbol{\mu}$ and the variance $\boldsymbol{\sigma}$, to set the threshold. That is, the threshold T is calculated as the two standard deviations $2\boldsymbol{\sigma}$ from the mean $\boldsymbol{\mu}$ of the anomaly values as follows:

$$T = \boldsymbol{\mu} + A\boldsymbol{\sigma}. \tag{15}$$

We set $A = 2$ and consider events with a higher anomaly score than T to be unusual. However, to detect more extreme anomalies, a larger A value can be used.

5 Experiments and Results

We first describe our data set and the experimental setup. We then performed experiments to answer the following questions:

- Ablation Study: How does the error change due to SVD truncation sizes and Tucker ranks in our proposed BID-Tucker?
- Scalability Test: How scalable is our proposed method compared to D-Tucker and vanilla Tucker decomposition?
- Application: What are the spatial and temporal properties of the air quality tensor that can be revealed by our proposed BID-Tucker?

5.1 Data Set and Experimental Setting

First, we describe the data preparation and experimental setup.

Air Quality Stream Data: We used Korean air quality data[1] from 2018.01.01 to 2022.12.31 (41616-time points) to construct a 3^{rd}-order temporal tensor for various spatial and temporal analyses. The Air Korea Air Quality Center measures and records particulate matter and air pollutants every hour from widely covered regions of South Korea, including remote islands. There are a total of 629 local monitoring stations with six types of measurements. The six types of measurements are ultrafine particulate matter (PM2.5; μgm^2(1 h)), fine particulate matter (PM10; μgm^2(1 h)), ozone (O_3; parts per million (ppm)), nitrogen dioxide (NO_2; ppm), carbon monoxide (CO; ppm), and sulfur dioxide (SO_2; ppm).

Constructing Air Quality Tensor: After obtaining the raw air quality data, we constructed a spatiotemporal tensor.

First, a min-max normalization was performed for each measurement type after missing values were handled. Missing value handling is important in Tucker decomposition methods that use SVDs, where each input cell value is used to find the singular vectors and values. A small number of missing values can either be filled with zero or interpolated with mean values. When a large amount of data is missing. Then the resulting singular vectors and values may be misleading. Thus, for stations with less than 10% missing values, we filled the missing values with the average value of the corresponding normalized air quality measurement type for the corresponding station. Stations with more than 10% missing values were removed, leaving 316 stations.

We then constructed a 3^{rd} order tensor, where the modes are ($time$, $location$, $type$). Overall, the shape of the obtained ($time$, $location$, $type$) indexed air quality tensor was $(41616, 316, 6)$. In the

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

experiment, we assumed that the tensor blocks are generated weekly for each hour ($7 * 24 = 168$), so the incoming blocks were set to tensors with a shape of $(168, 316, 6)$.

Experiment Settings: The study was conducted on a 12-core Intel(R) Core(TM) i7-6850K CPU @ 3.60 GHz. The following libraries were used with Python version 3.10: NumPy version 1.23.5, scikit-learn version 1.2.1, Pytorch 2.0.0, and Tensorly 0.8.1.

5.2 Ablation Study

There are two hyperparameters in the BID-Tucker: SVD truncation size and Tucker ranks. In the first ablation study, the SVD truncation size was varied and the corresponding normalized reconstruction error was measured to observe the effect of the SVD truncation size on the overall reconstruction error. In the second ablation study, Tucker ranks were varied, and normalized reconstruction errors were measured to evaluate the effect of rank sizes on overall reconstruction error. The normalized reconstruction error was calculated as $||\mathcal{X}_{org} - \mathcal{X}_{recon}||_F / ||\mathcal{X}_{org}||_F$. Again, we assumed that the air quality tensor of $(41616, 316, 6)$ comes in a tensor stream of the form $(168, 316, 6)$.

Fig. 2 (left) in the blue straight line shows the normalized reconstruction error as the SVD truncation sizes increase from 5 to 50 in the storage phase of the proposed BID-Tucker. In this test, the Tucker rank was set at $(30, 30, 6)$. The normalized error started to decrease slowly after the size of 20. Unless otherwise specified, the SVD truncation size was set to 20 in the remaining experiments.

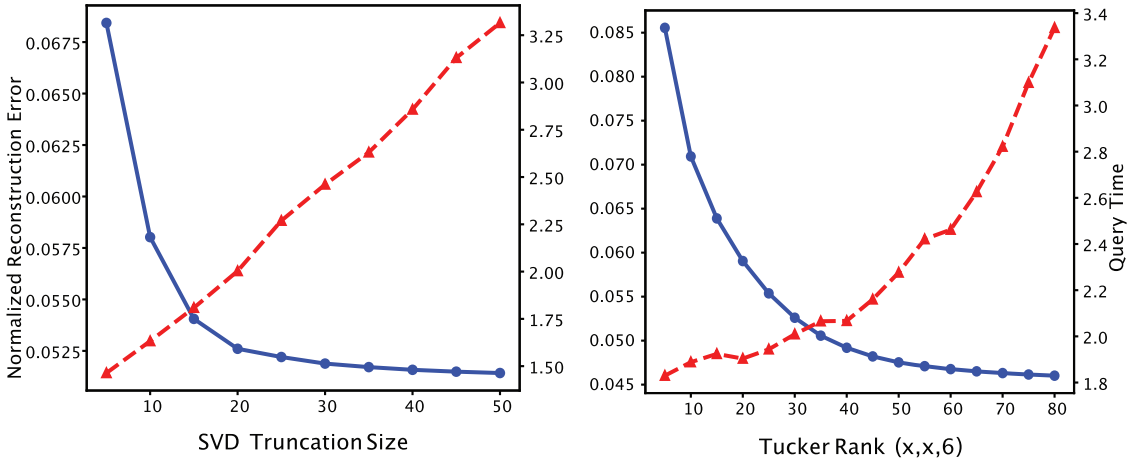

Figure 2: Normalized reconstruction error in blue solid lines and query processing time in red dotted lines over SVD truncation size with tensor ranks $(30, 30, 6)$ (left) and tensor rank in the form of $(x, x, 6)$ with SVD truncation size 20 (right)

The Fig. 2 (right) in the blue straight line shows the normalized reconstruction error for different Tucker rank tuples. The tensor ranks for the *time* and *location* modes were set equal. The rank for the *type* mode was set to 6. The ranks were varied from $(5, 5, 6)$ to $(80, 80, 6)$ in increments of 5. A rank size of 30 showed a low error for the air quality tensor. Unless otherwise specified, the Tucker ranks were set to $(30, 30, 6)$ in the remaining experiments.

5.3 Scalability Test

In the scalability test, we first tested the scalability according to the SVD truncation size and the Tucker rank. Fig. 2 (left) in the red dotted lines shows the query processing time as the SVD truncation

sizes increase from 5 to 50. The time complexity increased linearly as the SVD truncation size increased. Fig. 2 (right) in the red dotted lines shows the query processing time for different Tucker ranks. The Tucker ranks for the *time* and *location* modes were set equal, and the rank for the *type* modes was set to 6. The ranks were varied from (5, 5, 6) to (80, 80, 6), increasing by 5. The processing time increased in a near quadratic form which is also evident from the complexity analysis (Section 3.3) that shows the complexity to be proportional to the J^2 where J is the Tucker rank.

In the second part of the scalability test, we answer the question of how scalable our proposed BID-Tucker is compared to D-Tucker and regular Tucker decomposition. We varied the input tensor sizes and computed the error and query processing time for the proposed BID-Tucker, D-Tucker, and HOOI-based vanilla Tucker decomposition. Fig. 3 shows the normalized reconstruction error on the left and the query processing time on the right. We can see from the left plot that all three methods have similar reconstruction errors. However, as we can see from Fig. 3 (left) that the error differences are small, less than 0.001. With the comparable loss, we then looked at how scalable each method is to the size of the input tensor.

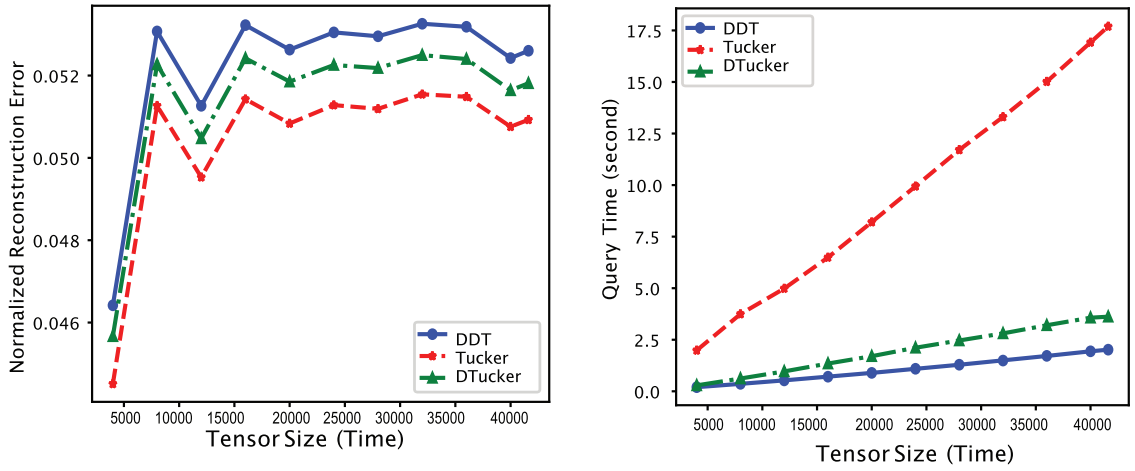

Figure 3: Comparison of normalized reconstruction error (left) and query processing time (right) of dynamically dense tucker decomposition (DDT) with D-tucker and HOOI-based vanilla tucker decomposition over different tensor sizes. The x axis is the length of the *time* mode for the input tensor, where the *location* mode dimension is fixed to 316 and the *type* mode dimension is fixed to 6

For a fair comparison, we included the processing time for both the storage and query phases and set the query range to the full range for our BID-Tucker. Fig. 3 (right) shows that our proposed BID-Tucker significantly outperformed vanilla Tucker and showed a lower growth rate compared to D-Tucker. The lower growth rate compared to D-Tucker comes from efficiency grained by block incremental SVD used in the storage phase.

Overall, our proposed method scales better than the compared methods, with a tolerable error difference.

5.4 Modeling Annual Trends of Korean Air Quality

We describe our analysis results for the Korean air quality data. First, we used the time mode factor matrix to experiment with whether we could detect unusual air quality. Then, we analyzed whether our method could detect regional similarities in air quality by analyzing the location mode factor matrix.

Temporal Modeling and Anomaly Detection: We clustered the time mode factor matrix of the decomposed air quality tensor for modeling the temporal trends and detecting unusual air quality events in Korean air quality. Specifically, we used the time mode factor matrix $A_T \in \mathbb{R}^{(41616\times30)}$ over the entire data period from the year 2018 to 2022 over the latent dimensions to perform k-means clustering. The optimal $k = 3$ was found by the elbow method, which plots the within-cluster sum of squares (WCSS) against the cluster size k (Fig. 4 (left)). Fig. 4 (right) shows the clustering result visualized by t-SNE (t-distributed stochastic neighbor embedding), where each color corresponds to a cluster.

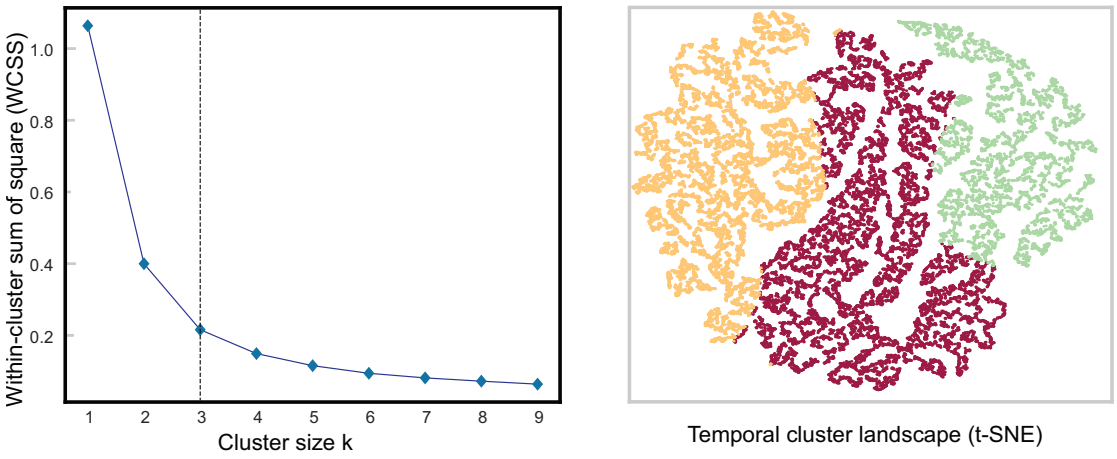

Figure 4: Temporal analysis of air quality data using k-means. The elbow method is used and plotted on the left to find optimal clusters of size three, the color-coded cluster distribution is plotted using t-SNE on the right

The three clusters map well to Korea's four seasons, each of which has its own characteristics. Specifically, Cluster 1 (yellow) maps mainly to summer periods when temperatures and humidity are high nationwide. Ozone alerts are often issued during this time. Cluster 2 (red) mostly maps to the spring and fall seasons. Yellow dust and fine particle alerts are common during this time. Cluster 3 (purple) was mainly associated with the winter seasons. This period is characterized by cold, dry weather across the country, with occasional fine particle alerts.

To detect unusual air quality over time, we calculated anomaly scores as described in Section 4 using the Euclidean distance to the nearest cluster centroid of the time mode factor matrix. We considered anomaly scores above the *mean* $+2\times$ *standard deviation* to be unusual events. The vertical line in Fig. 5 corresponds to the threshold. We took a closer look at some of the unusual time points to see if there were any notable events in the news that could have led to the time point having a high anomaly score.

We found two interesting items, chronic ozone alerts issued during the late spring and early summer periods and large fire events. The period from May to August between the years 2019 to 2022 contains several days with unusually high ozone levels[2] . The time of high ozone concentration was between 13:00 and 17:00. The high ozone concentration during these periods occurred between

[2]List of days and times of ozone alerts issued: 2019-05-23 25, 2019-06-19 21, 2019-08-02 05, 2019-08-18 20, 2020-06-06 09, 2020-07-16 17, 2020-08-18 21, 2021-05-13 19, 2021-06-05 09, 2021-06-20 25, 2021-07-20 24, 2022-05-23 25, 2022-06-20 22, 2022-07-10 16, 2022-07-25 26.

13:00 and 17:00, and the concentrations were two to three times higher than on normal days. These chronic events are marked in red in Fig. 5.

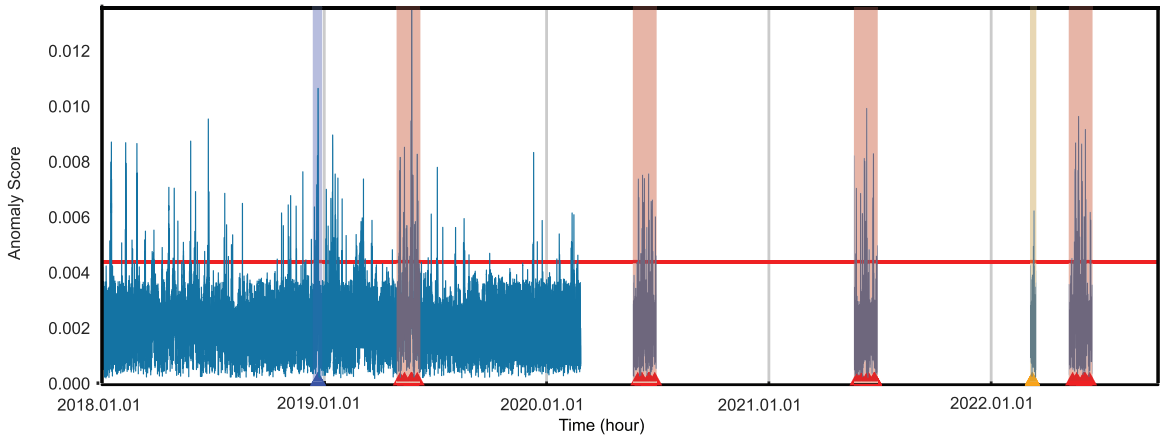

Figure 5: Temporal anomaly score plot with threshold and selected events. The x axis is the time points in chronological order from 2018-01-01 to 2022-12-31. The vertical line is the threshold at $t = mean + 2 \times standard deviation$. Red marks denote chronic high ozone concentration events. Blue marks denote the two construction fire events. Organ marks denote several forest fire events

For fire events, we found several. Two marked cases are as follows: large construction fires and forest fire events. On December 28, 2018, there were two construction fires: the Jeonbuk sewage treatment plant fire on 2018-12-28 between 7:00 and 8:00, and the Busan factory fire on 2018-12-28 between 1:00 and 3:00. The anomaly values mean the effect of the fires on air quality lasted for a few days after the events. These events are marked in blue in Fig. 5. We also confirmed that large forest fires were detectable by the anomaly values. During the period from March 04, 2022 to March 11, 2022, there were several small and large fire events throughout the east coast of South Korea. The forest fire events are marked in orange in Fig. 5.

In addition, Fig. 5 shows that there is a difference in the pattern of anomaly values before and after the COVID-19 pandemic break. Before the pandemic break in December 2019, there were much more high peaks of anomaly air quality than during the pandemic. We speculate that this is related to the decrease in factory and vehicle utilization rates in and around nations during the pandemic.

Spatial Modeling: For the spatial modeling based on the air quality tensor, we performed K-means clustering on the location mode factor matrix $\mathbf{A}_L \in \mathbb{R}^{316\times30}$. Similar to the temporal analysis, we selected the optimal $k = 3$ using the elbow method (Fig. 6a) and visualized the clustering result using t-SNE (Fig. 6b).

For each of the three clusters, we found distinct local similarities, which we list below. Cluster 1 (yellow) corresponded to the metropolitan areas, including Seoul and Incheon. Cluster 2 (purple) corresponded to the eastern half of South Korea, which is known to have better overall air quality throughout the year. The eastern half is mostly mountainous. Cluster 3 (red) corresponded to the western half of South Korea, which is known to have poor overall air quality compared to the eastern half, even in rural areas. The western half has fewer mountains and more plains.

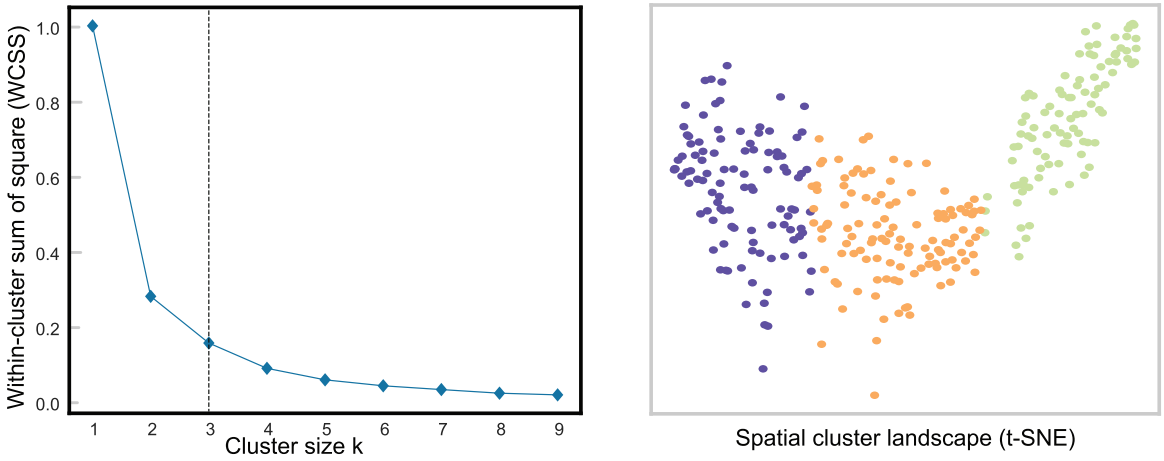

Figure 6: Spatial analysis of air quality data using k-means. The elbow method is used and plotted on the left to find optimal clusters of size three, the color-coded cluster distribution is plotted using t-SNE on the right

6 Conclusion

Recognizing the need for efficient storage of spatiotemporal stream data in dynamic modeling, we proposed Block Incremental Dense Tucker Decomposition (BID-Tucker), which can provide efficient storage and on-demand Tucker decomposition results. Our BID-Tucker can be applied to any tensor data that comes in tensor blocks where the dimensions of modes stay the same except for the time mode. We also assume only partial time-ranged data needs to be analyzed and modeled. Our BID-Tucker stores the singular value decomposition (SVD) results of each sliced matrix of a tensor block instead of the original tensor. The SVD results of each time block are concatenated by block incremental SVD [13] and used to initialize the Tucker decomposition [12]. We have applied our BID-Tucker to the Korean Air Quality data for modeling the air quality trends. When applying it to the air quality tensor, we provided an approach to find the optimal SVD truncation size and Tucker rank. We used the same optimal setting to compare and show benefits in the error rate and scalability of our BID-Tucker to that of D-Tucker, which is an efficient Tucker decomposition method that our BID-Tucker extends, and the vanilla Tucker method, which is known to give the best error although not efficient. We further applied our BID-Tucker on the air quality tensor to model the air quality trends and detect unusual events. Also, we were able to verify that there are spatial similarities in the air qualities. Overall, we conclude that our proposed BID-Tucker was able to provide efficient storage and Tucker decomposition results on-demand for time block tensors. Although we only applied the BID-Tucker to the air quality tensor, it can be applied to various multi-dimensional spatiotemporal data, and the post-decomposition analysis process can be used for modeling spatial and temporal trends.

Acknowledgement: The authors wish to express their appreciation to the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-00369) and by the National Research Foundation of Korea Grant funded by the Korean government (2018R1A5A1060031, 2022R1F1A1065664).

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: S. Lee, L. Sael; data collection: H. Moon; analysis and interpretation of results: S. Lee, H. Moon, L. Sael; draft manuscript preparation: S. Lee, H. Moon, L. Sael. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The Korean Air Quality Data has been obtained through the Air Korea Insitute portal at https://www.airkorea.or.kr/. Our codes for BID-Tucker and D-Tucker, written in Python, are provided in the GitHub https://github.com/Ajou-DILab/BID-Tucker.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Jeon, I., Papalexakis, E. E., Kang, U., Faloutsos, C. (2015). HaTen2: Billion-scale tensor decompositions. *Proceedings of the 31st IEEE International Conference on Data Engineering*, Seoul, South Korea, IEEE.
2. Park, N., Jeon, B., Lee, J., Kang, U. (2016). Bigtensor: Mining billion-scale tensor made easy. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, Indianapolis, IN, USA.
3. Choi, D., Jang, J. G., Kang, U. (2019). S3CMTF: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization. *PLoS One, 14(6),* 1–20.
4. Jang, J. G., Moonjeong Park, J. L., Sael, L. (2022). Large-scale tucker tensor factorization for sparse and accurate decomposition. *Journal of Supercomputing, 78,* 17992–18022.
5. Gujral, E., Pasricha, R., Papalexakis, E. E. (2018). Sambaten: Sampling-based batch incremental tensor decomposition. *Proceedings of the 2018 SIAM International Conference on Data Mining*, San Diego, California, USA.
6. Kwon, T., Park, I., Lee, D., Shin, K. (2021). Slicenstitch: Continuous cp decomposition of sparse tensor streams. *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, Chania, Greece, IEEE.
7. Smith, S., Huang, K., Sidiropoulos, N. D., Karypis, G. (2018). Streaming tensor factorization for infinite data sources. *Proceedings of the 2018 SIAM International Conference on Data Mining*, San Diego, CA, USA.
8. Zhou, S., Vinh, N. X., Bailey, J., Jia, Y., Davidson, I. (2016). Accelerating online cp decompositions for higher order tensors. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA.
9. Kolda, T. G., Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review, 51(3),* 455–500.
10. Chachlakis, D. G., Dhanaraj, M., Prater-Bennette, A., Markopoulos, P. P. (2021). Dynamic l1-norm tucker tensor decomposition. *IEEE Journal of Selected Topics in Signal Processing, 15(3),* 587–602.
11. Jang, J. G., Kang, U. (2023). Static and streaming tucker decomposition for dense tensors. *ACM Transactions on Knowledge Discovery from Data, 17(5),* 1–34.
12. Jang, J. G., Kang, U. (2020). D-tucker: Fast and memory-efficient tucker decomposition for dense tensors. *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, Dallas, Texas, USA.
13. Jinhong Jung, L. S. (2020). Fast and accurate pseudoinverse with sparse matrix reordering and incremental approach. *Machine Learning, 109,* 2333–2347.
14. Vannieuwenhoven, N., Vandebril, R., Meerbergen, K. (2012). A new truncation strategy for the higher-order singular value decomposition. *SIAM Journal on Scientific Computing, 34(2),* A1027–A1052.
15. Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M. (2008). A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis, 25(3),* 335–366.