# Fast and Memory-Efficient Tucker Decomposition for Answering Diverse Time Range Queries

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

U Kang
Seoul National University
Republic of Korea
ukang@snu.ac.kr

## ABSTRACT

Given a temporal dense tensor and an arbitrary time range, how can we efficiently obtain latent factors in the range? Tucker decomposition is a fundamental tool for analyzing dense tensors to discover hidden factors, and has been exploited in many data mining applications. However, existing decomposition methods do not provide the functionality to analyze a specific range of a temporal tensor. The existing methods are one-off, with the main focus on performing Tucker decomposition once for a whole input tensor. Although a few existing methods with a preprocessing phase can deal with a time range query, they are still time-consuming and suffer from low accuracy. In this paper, we propose Zoom-Tucker, a fast and memory-efficient Tucker decomposition method for finding hidden factors of temporal tensor data in an arbitrary time range. Zoom-Tucker fully exploits block structure to compress a given tensor, supporting an efficient query and capturing local information. Zoom-Tucker answers diverse time range queries quickly and memory-efficiently, by elaborately decoupling the preprocessed results included in the range and carefully determining the order of computations. We demonstrate that Zoom-Tucker is up to 171.9× faster and requires up to 230× less space than existing methods while providing comparable accuracy.

## CCS CONCEPTS

• **Computing methodologies** → **Factorization methods**.

## KEYWORDS

Tucker decomposition, time range query, efficiency

**ACM Reference Format:**
Jun-Gi Jang and U Kang. 2021. Fast and Memory-Efficient Tucker Decomposition for Answering Diverse Time Range Queries . In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467290

## 1 INTRODUCTION

Given a temporal dense tensor and a time range (e.g., January - March 2019), how can we efficiently analyze the tensor in the given time range? Many real-world data including stock data, video data, and traffic volume data are represented as temporal dense tensors. Tensor decomposition has played an important role in various applications including data clustering [4, 10], concept discovery [1, 13, 14], dimensionality reduction [16, 36], anomaly detection [18], and link prediction [19, 24]. Tucker decomposition, one of the tensor decomposition methods, has been recognized as a crucial tool for discovering latent factors and detecting relations between them.

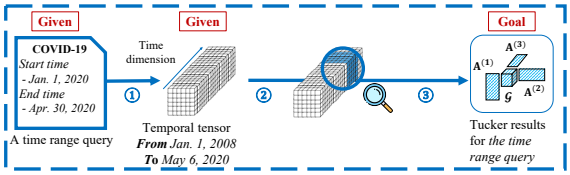


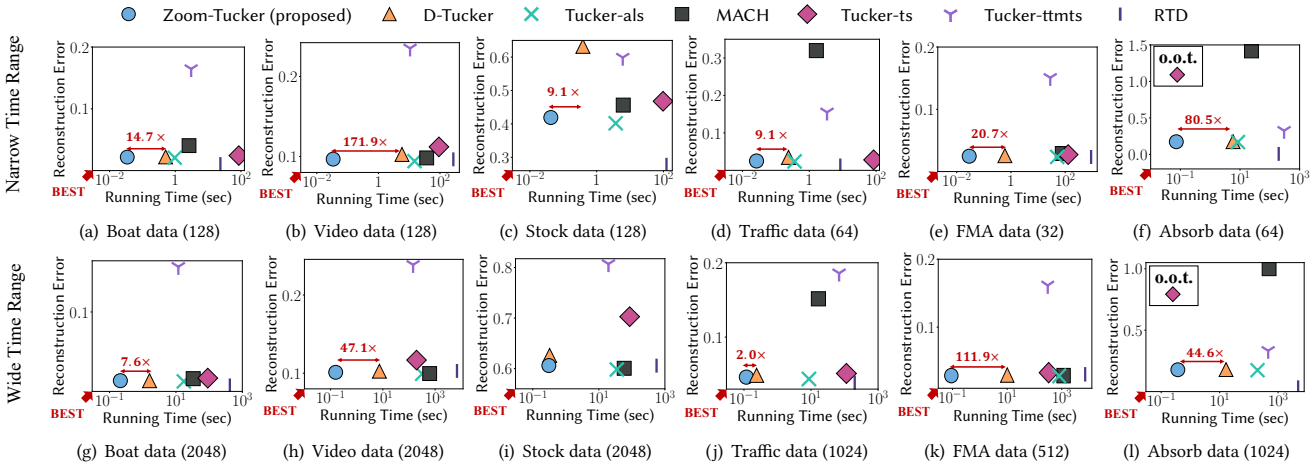

**Figure 1: Given a temporal tensor and a user-provided time range (start time and end time) query, the goal of the time-ranged Tucker decomposition is to find the patterns of the temporal tensor at the range using Tucker decomposition.**

In practice, we analyze a given temporal tensor from various perspectives. Assume a user is interested in investigating patterns of various time ranges using Tucker decomposition. Given a temporal tensor and a user-provided time range (start time and end time) query, our goal is to find the patterns of the temporal tensor at the range using Tucker decomposition. For example, given a temporal tensor including matrices collected between Jan. 1, 2008 to May 6, 2020, a user may be interested in Tucker decomposition of a subrange between Jan. 1, 2020 to April 30, 2020 (see Figure 1). Since Tucker decomposition generates factor matrices and a core tensor to accurately approximate an input tensor, answering time range queries, (i.e., performing Tucker decomposition of different sub-tensors) yields different Tucker results. However, conventional Tucker decomposition methods [5, 20, 25] based on Alternating Least Square (ALS) is not appropriate for answering diverse time range queries since they target performing Tucker decomposition once for a given tensor; the methods require a high computational cost and large storage space since they need to perform Tucker decomposition of the sub-tensor included in a time range query from scratch, every time the query is given. Due to this limitation, the existing methods are not efficient in exploring diverse time ranges for a given temporal tensor.

A few methods [12, 35] with a preprocessing phase can be adapted to the time range query problem; before the query phase, they preprocess a given tensor, and perform Tucker decomposition with the preprocessed tensor for each time range query. However, they suffer from an accuracy issue for narrow time ranges since preprocessed results are tailored for performing Tucker decomposition of the whole given temporal tensor. The results fail to capture local patterns that appear only in a specific range.


*KDD '21, August 14–18, 2021, Virtual Event, Singapore*

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467290




(a) Boat data (128) (b) Video data (128) (c) Stock data (128) (d) Traffic data (64) (e) FMA data (32) (f) Absorb data (64)

(g) Boat data (2048) (h) Video data (2048) (i) Stock data (2048) (j) Traffic data (1024) (k) FMA data (512) (l) Absorb data (1024)

**Figure 2: Trade-off between query time and reconstruction error of Zoom-Tucker and competitors, for narrow (a-f) and wide (g-l) time range queries. o.o.t.: out of time (takes more than 20,000 seconds). Numbers after the data name represent the length of time ranges; e.g., (128) means the length of a time range is 128 timesteps. Zoom-Tucker is closest to the best point with the fastest query speed and the lowest reconstruction error.**

**Table 1: Symbol description.**

| Symbol | Description |
|---|---|
| $\mathcal{X}$ | temporal tensor ($\in I_1 \times \ldots \times I_N$) |
| $I_n$ & $J_n$ | dimensionality of the $n$-th mode of $\mathcal{X}$ and $\mathcal{G}$ |
| $b$ | block size |
| $t_s$ & $t_e$ | starting and ending points of time range query |
| $[t_s, t_e]$ | time range of a query |
| $\mathcal{X}^{<i>}$ | $i$-th temporal block tensor ($\in I_1 \times \ldots I_{N-1} \times b$) |
| $(\mathbf{A}^{<i>})^{(k)}$ | $k$-th factor matrix of $i$-th temporal block tensor |
| $\mathcal{G}^{<i>}$ | core tensor of $i$-th temporal block tensor |
| $\tilde{\mathcal{X}}$ | temporal tensor obtained in the time range $[t_s, t_e]$ |
| $\tilde{\mathbf{A}}^{(k)}$ | $k$-th factor matrix of time range query $[t_s, t_e]$ |
| $\tilde{\mathcal{G}}$ | core tensor of time range query $[t_s, t_e]$ |
| $S$ | index of temporal block tensor corresponding to $t_s$ |
| $E$ | index of temporal block tensor corresponding to $t_e$ |
| $\otimes$ | Kronecker product |
| $\times_n$ | $n$-mode product |

In this paper, we propose Zoom-Tucker (Zoomable Tucker decomposition), a fast and memory-efficient Tucker decomposition method to analyze a temporal tensor for diverse time ranges. Zoom-Tucker enables us to discover local patterns in a narrow time range (zoom-in), or global patterns in a wider time range (zoom-out). Zoom-Tucker consists of two phases: the preprocessing phase and the query phase. The preprocessing phase of Zoom-Tucker exploits block structure to lay the groundwork in achieving an efficient query phase and capturing local information. In the query phase, Zoom-Tucker addresses the high computational cost and space cost by elaborately decoupling block results and carefully determining the order of computation. Thanks to these ideas, Zoom-Tucker answers an arbitrary time range query with higher efficiency than existing methods. Through extensive experiments, we demonstrate the effectiveness and efficiency of our method compared to other methods. The main contributions of this paper are as follows:

- **Algorithm.** We propose Zoom-Tucker, a fast and memory-efficient Tucker decomposition method for answering diverse time range queries.
- **Analysis.** We provide both time and space complexities for the preprocessing and query phases of Zoom-Tucker.
- **Experiment.** Experimental results show that Zoom-Tucker answers time range queries up to 171.9× faster and requires up to 230× less space than other methods while providing comparable accuracy, as shown in Figures 2 and 6.
- **Discovery.** Thanks to Zoom-Tucker, we discover anomalous ranges and trend changes in Stock dataset (Figures 8 and 9).

The code of our method and datasets are available at **https://datalab.snu.ac.kr/zoomtucker**.

## 2 PRELIMINARIES

We describe preliminaries on tensor operations and Tucker decomposition, and then define the problem addressed in this paper (Section 2.3). The symbols we use in this paper are described in Table 1.

### 2.1 Tensor and Its Operation

A tensor is a multi-dimensional array. Each dimension of a tensor is called *mode*. The length of each mode is called dimensionality and denoted by $I_1, \cdots, I_N$. In this paper, a vector, a matrix, and an $N$-mode tensor are denoted by the boldface lower case (e.g. $\mathbf{a}$), boldface capitals (e.g. $\mathbf{A}$), and boldface Euler script capital (e.g. $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$), respectively. Key operations for tensor include Frobenius norm, Kronecker product, mode-$n$ matricization, and $n$-mode product. We refer the reader to [17] for their definitions.

### 2.2 Tucker Decomposition

Tucker decomposition transforms an $N$-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ and factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, N$. Factor matrices $\mathbf{A}^{(n)}$ are column-orthogonal, and a core tensor $\mathcal{G}$ is small and dense. Each factor matrix $\mathbf{A}^{(n)}$ represents the latent features of the $n$-th mode of $\mathcal{X}$, and each element of a core tensor $\mathcal{G}$ is the weight of the relation composed of columns of factor matrices. Given a tensor $\mathcal{X}$, the goal of Tucker decomposition is to obtain factor matrices $\mathbf{A}^{(n)}$ and the core tensor $\mathcal{G}$ by minimizing $\|\mathcal{X} - \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}\|_F^2$ as shown in the following equations:

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)} \Leftrightarrow \mathbf{X}_{(n)} \approx \mathbf{A}^{(n)} \mathbf{G}_{(n)} (\otimes_{k \neq n}^{N} \mathbf{A}^{(k)T}) \quad (1)$$

Note that $\mathbf{X}_{(n)}$ indicates the mode-$n$ matricized version of $\mathcal{X}$, $\mathbf{G}_{(n)}$ indicates the mode-$n$ matricized version of $\mathcal{G}$, and $(\otimes_{k \neq n}^{N} \mathbf{A}^{(k)T})$ indicates performing the entire Kronecker product of $\mathbf{A}^{(k)T}$ in descending order for $k = N, ..., n+1, n-1, ..., 1$.

ALS (Alternating Least Square) is a common approach for Tucker decomposition as described in Appendix A. ALS approach iteratively updates a factor matrix of a mode while fixing all factor matrices of other modes. For updating each factor matrix $\mathbf{A}^{(n)}$, a dominant operation is to compute $n$-mode products between an input tensor $\mathcal{X}$ ($\in I_1 \times ... \times I_N$) and factor matrices $\mathbf{A}^{(k)}$ ($\in I_k \times J_k$) for $k = N, ..., n+1, n-1, ..., 1$ (line 4 of Algorithm 3). Computing $\mathbf{X}_{(n)}(\otimes_{k \neq n}^{N} \mathbf{A}^{(k)})$, the mode-$n$ matricized version of the dominant operation, takes $\mathcal{O}(J \prod_{i=1}^{N} I_i)$ time where $\mathbf{X}_{(n)}$ is the mode-$n$ matricized version of $\mathcal{X}$.

## 2.3 Problem Definition

We describe the formal definition of the time range query problem as follows:

Problem 1 (Time Range Query on Temporal Tensor).
***Given:*** *a temporal dense tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ and a time range $[t_s, t_e]$ where $I_N$ is the length of the time dimension, and $I_n$ is the dimensionality of mode-n for $n = 1, ..., N-1$,*
***Find:*** *the Tucker results of the sub-tensor $\tilde{\mathcal{X}}$ of $\mathcal{X}$ in the time range $[t_s, t_e]$ efficiently. The Tucker result includes factor matrices $\tilde{\mathbf{A}}^{(1)}$, ..., $\tilde{\mathbf{A}}^{(N)}$, and core tensor $\tilde{\mathcal{G}}$.*

To address the time range query problem, a method should efficiently handle various time range queries. Given an arbitrary time range query, existing methods [5, 20, 25] performing Tucker decomposition from scratch requires a high computational cost and large space cost. Compared to the aforementioned methods, Tucker decomposition methods [12, 35] with a preprocessing phase save time and space costs in that they allow us to compress a whole tensor before a query phase, and then perform Tucker decomposition of a sub-tensor corresponding to a given time range query by exploiting the compressed tensor instead of the input tensor. However, they are still unsatisfactory in terms of time, space, and accuracy for the time range query problem since they are tailored for performing Tucker decomposition of only the whole tensor once.

# 3 PROPOSED METHOD

In this section, we propose Zoom-Tucker, a novel method for extracting key patterns of a temporal tensor in an arbitrary time range. The following challenges need to be tackled:

- C1 **Dealing with various time range queries.** Each user deals with different time ranges or a user analyze patterns for various time ranges. How can we preprocess a temporal tensor to deal with various time ranges?
- C2 **Minimizing computational cost.** Tucker decomposition requires a high computational cost. How can we quickly perform Tucker decomposition for a given time range query?
- C3 **Minimizing intermediate data.** Imprudent computation for Tucker decomposition provokes huge intermediate data. How can we avoid generating huge intermediate data?

We address the challenges with the following main ideas:

**Algorithm 1:** Preprocessing phase of Zoom-Tucker

**Input:** temporal tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1} \times I_N}$
**Output:** result sets $\mathcal{C}_n$ for $n = 1, \ldots, N+1$
**Parameters:** block size $b$
1: compute the number $B = \lceil \frac{I_N}{b} \rceil$ of blocks
2: split $\mathcal{X}$ into block tensors $\mathcal{X}^{<i>} \in \mathbb{R}^{I_1 \times \ldots \times b}$ for $i = 1, \ldots, B$
3: **for** $i \leftarrow 1$ to $B$ **do**
4: perform Tucker decomposition of $\mathcal{X}_i \approx \mathcal{G}^{<i>} \times_1 (\mathbf{A}^{<i>})^{(1)} \cdots \times_N (\mathbf{A}^{<i>})^{(N)}$
5: store each factor matrices $(\mathbf{A}^{<i>})^{(n)}$ in the results set $\mathcal{C}_n$, for $n = 1, \ldots, N$
6: store core tensor $\mathcal{G}^{<i>}$ in the result set $\mathcal{C}_{N+1}$
7: **end for**

- I1 **Exploiting block structure** enables a query phase to decrease the number of operations and memory requirements, while capturing local information.
- I2 **Elaborately decoupling block results** decreases the computational cost of Tucker decomposition for a tensor obtained in a given time range.
- I3 **Carefully determining the order of computation** minimizes intermediate data generation while avoiding redundant computation.

Zoom-Tucker efficiently computes Tucker decomposition for various time range queries. Zoom-Tucker consists of two phases: the preprocessing phase and the query phase. The preprocessing phase is computed once for a given temporal tensor, while the query phase is computed using the results of the preprocessing phase for each time range query. Zoom-Tucker compresses a given tensor block by block along the time dimension in the preprocessing phase. Zoom-Tucker performs Tucker decomposition for each block. In the query phase, Zoom-Tucker performs Tucker decomposition for each time range query by 1) adjusting the first and the last blocks included in the time range to fit the range and 2) carefully stitching the block results in the time range.

## 3.1 Preprocessing Phase

The objective of the preprocessing phase is to manipulate a given temporal tensor for an efficient query phase. In the query phase, performing Tucker decomposition from scratch requires high computational cost and large space cost as the number of queries increases. To avoid it, compressing a given tensor is inevitable to provide fast processing in the query phase. Additionally, we consider that compressed results need to contain local patterns that appear only in specific ranges. The preprocessing phase of existing Tucker decomposition methods [12, 25, 35] fails to support high efficiency of the query phase while maintaining local patterns. Then, how can we compress a given tensor to deal with various time range queries? Our main idea is to exploit a block structure: 1) carefully designating the form of a block, and 2) selecting a compression approach for each block. In this paper, we 1) split a given temporal tensor into sub-tensors along the time dimension, and 2) leverage Tucker decomposition for each sub-tensor. The idea allows Zoom-Tucker to support an efficient query phase and capture local patterns. Additionally, the preprocessing phase is extensible for new incoming tensors by performing Tucker decomposition of them.

To capture local information, we split a given tensor along the time dimension. Let the reconstruction error at each timestep $t$ be

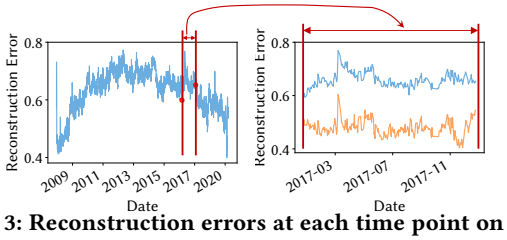



**Figure 3: Reconstruction errors at each time point on Stock dataset. The blue line presents reconstruction errors computed from a whole temporal tensor, while the orange line describes reconstruction errors computed from a sub-tensor in a range. Performing Tucker decomposition from a sub-tensor provides relatively low reconstruction errors.**

measured by performing Tucker decomposition. The reconstruction error is defined as $\frac{\|\mathcal{X}(t)-\hat{\mathcal{X}}(t)\|_F^2}{\|\mathcal{X}(t)\|_F^2}$ where $\mathcal{X}(t)$ is an input sub-tensor obtained at each timestep $t$ and $\hat{\mathcal{X}}(t)$ is the sub-tensor at timestep $t$ reconstructed from Tucker results. Figure 3 shows the reconstruction errors of Stock dataset at each time point. Given a sub-tensor in a range that has relatively high errors, performing Tucker decomposition of the sub-tensor (orange line in Figure 3) provides lower errors than the preceding result computed from a whole temporal tensor (blue line in Figure 3). This observation implies that decomposing a sub-tensor allows us to capture local information, leading to low errors. Based on the observation, we construct sub-tensors by splitting a temporal tensor along the time dimension and perform Tucker decomposition of each sub-tensor. It provides lower error than performing Tucker decomposition of a whole tensor on all the timesteps, by capturing local information.

To support an efficient query phase, we store the Tucker decomposition results of sub-tensors. There are the two benefits to leveraging Tucker decomposition in the preprocessing phase: 1) saving the space cost due to the small preprocessed results compared to the given tensor, and 2) enabling the query phase to exploit *the mixed-product property* applicable to mixing matrix multiplication and Kronecker product, i.e., $(\mathbf{A}^T \otimes \mathbf{B}^T)(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A}^T\mathbf{C} \otimes \mathbf{B}^T\mathbf{D})$. Computing $(\mathbf{A}^T\mathbf{C} \otimes \mathbf{B}^T\mathbf{D})$ requires less costs than computing $(\mathbf{A}^T \otimes \mathbf{B}^T)(\mathbf{C} \otimes \mathbf{D})$ when the size of the four matrices is $I \times J$ and $I >> J$. The reason is that the size of $\mathbf{A}^T\mathbf{C}$ and $\mathbf{B}^T\mathbf{D}$ is only $J \times J$ while the size of $(\mathbf{A}^T \otimes \mathbf{B}^T)$ and $(\mathbf{C} \otimes \mathbf{D})$ is $J^2 \times I^2$ and $I^2 \times J^2$, respectively. We further present the exploitation of this property to achieve high efficiency of the query phase in Sections 3.2.3 and 3.2.4.

Figure 4 presents an overview of the preprocessing phase. Without loss of generality, we assume that the temporal mode is the last mode ($N$th mode). We express a given tensor $\mathcal{X}$ as temporal block tensors $\mathcal{X}^{<i>} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1} \times b}$ for $i = 1, ..., \lceil \frac{I_N}{b} \rceil$ (line 2 in Algorithm 1) where $b$ is a block size and $I_N$ is the dimensionality of the time dimension. Then, we perform Tucker decomposition for each temporal block tensor $\mathcal{X}^{<i>}$ (line 4 in Algorithm 1), and store each factor matrix $(\mathbf{A}^{<i>})^{(n)}$ in a set $\mathcal{C}_n$ and the core tensor $\mathcal{G}^{<i>}$ in a set $\mathcal{C}_{N+1}$ (lines 5 and 6 in Algorithm 1). Since the preprocessing phase is computed once and affects errors of the query phase, this phase prefers an accurate but slow Tucker decomposition method rather than a fast but approximate Tucker decomposition one. Specifically, we use Tucker-ALS, which is stable and accurate, in this phase.

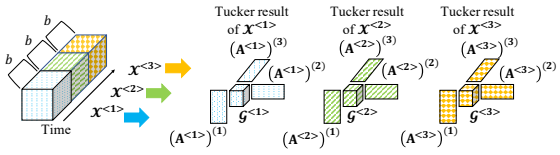



**Figure 4: Preprocessing phase of Zoom-Tucker.**

## 3.2 Query Phase

The objective of the query phase is to efficiently compute Tucker decomposition for a given time range $[t_s, t_e]$. The query phase of Zoom-Tucker operates as follows:

- S1. Given a time range $[t_s, t_e]$, we load Tucker results (i.e., $\mathcal{G}^{<i>}$, $(\mathbf{A}^{<i>})^{(n)}$) of temporal block tensors $\mathcal{X}^{<i>}$ for $i = S, ..., E$ where $S = \lceil \frac{t_s}{b} \rceil$ and $E = \lceil \frac{t_e}{b} \rceil$ are the indices of the first and the last temporal block tensors including $t_s$ and $t_e$, respectively.
- S2. We adjust the Tucker results of $\mathcal{X}^{<S>}$ and $\mathcal{X}^{<E>}$ to fit the range since a part of them may not be within the given range.
- S3. Given the Tucker results of $\mathcal{X}^{<i>}$ for $i = S, .., E$ included in the range, Zoom-Tucker updates factor matrices by efficiently stitching the Tucker results.
- S4. After that, Zoom-Tucker updates the core tensor using factor matrices updated at Step S3 and the Tucker results.
- S5. Zoom-Tucker repeatedly performs Steps S3 and S4 until convergence.

The most important challenge of the efficient query phase is how to minimize the computational cost for updating factor matrices (Step S3) and the core tensor (Step S4) of the time range while minimizing the intermediate data. To tackle the challenge, our main ideas are to 1) elaborately decouple $\tilde{\mathbf{X}}_{(n)} \left(\otimes_{k \neq n}^{N} \tilde{\mathbf{A}}^{(k)T}\right)$ based on preprocessed results, and 2) carefully determine the order of computation. We first give an objective function and an update rule for the query phase (Section 3.2.1). Then, we describe how to achieve high efficiency of Zoom-Tucker in detail (Sections 3.2.2 to 3.2.4).

*3.2.1 Objective function and update rule.* In the query phase, our goal is to obtain factor matrices $\tilde{\mathbf{A}}^{(1)}$, ..., $\tilde{\mathbf{A}}^{(N)}$, and core tensor $\tilde{\mathcal{G}}$ for a given time range query $[t_s, t_e]$. The query phase of Zoom-Tucker alternately updates factor matrices, and core tensor as in ALS. We minimize the following objective function as mode-$n$ matricized form for a time range $[t_s, t_e]$:

$$L_{(n)} = \|\tilde{\mathbf{X}}_{(n)} - \tilde{\mathbf{A}}^{(n)}\tilde{\mathbf{G}}_{(n)}(\otimes_{k \neq n}^{N} \tilde{\mathbf{A}}^{(k)T})\|_F^2 \tag{2}$$

where $\tilde{\mathbf{X}}_{(n)}$ is the mode-$n$ matricized version of a tensor obtained in the time range $[t_s, t_e]$, and $\tilde{\mathbf{G}}_{(n)}$ is the mode-$n$ matricized version of $\tilde{\mathcal{G}}$. From the objective function (2), we derive the following update rule for $n$-th factor matrix (see the proof in Appendix B.1):

Lemma 1 (Update rule). *When fixing all but the n-th factor matrix, the following update rule for the n-th factor matrix minimizes the objective function* (2).

$$\tilde{\mathbf{A}}^{(n)} \leftarrow \tilde{\mathbf{X}}_{(n)} \left(\otimes_{k \neq n}^{N} \tilde{\mathbf{A}}^{(k)}\right) \tilde{\mathbf{G}}_{(n)}^T \left(\mathbf{C}^{(n)}\right)^{-1} \tag{3}$$

*where* $\mathbf{C}^{(n)} \in \mathbb{R}^{J_n \times J_n}$ *of the n-th mode is given by*

$$\mathbf{C}^{(n)} = \tilde{\mathbf{G}}_{(n)} \left(\otimes_{k \neq n}^{N} \tilde{\mathbf{A}}^{(k)T}\tilde{\mathbf{A}}^{(k)}\right) \tilde{\mathbf{G}}_{(n)}^T \qquad \square$$

**Algorithm 2:** Query phase of Zoom-Tucker

**Input:** a time range $[t_s, t_e]$, and Tucker result sets $\mathcal{C}_n$ for n = 1, …, $N+1$
**Output:** factor matrices $\tilde{\mathbf{A}}^{(n)}$ for $n = 1, .., N$, and core tensor $\tilde{\mathcal{G}}$
**Parameters:** tolerance $\epsilon$, and block size $b$

1: $S \leftarrow \lceil \frac{t_s}{b} \rceil$ and $E \leftarrow \lceil \frac{t_e}{b} \rceil$
2: load $(\mathbf{A}^{<i>})^{(k)}$ and $\mathcal{G}^{<i>}$ for $i = S, \ldots, E$ from $\mathcal{C}_k$ for $k = 1, \ldots, N+1$
3: obtain $(\bar{\mathbf{A}}^{<S>})^{(N)}$ and $(\bar{\mathbf{A}}^{<E>})^{(N)}$ by eliminating the rows of $(\mathbf{A}^{<S>})^{(N)}$ and $(\mathbf{A}^{<E>})^{(N)}$ excluded in the range
4: $(\bar{\mathbf{A}}^{<S>})^{(N)} \rightarrow \mathbf{Q}^{<S>}\mathbf{R}^{<S>}$, $(\bar{\mathbf{A}}^{<E>})^{(N)} \rightarrow \mathbf{Q}^{<E>}\mathbf{R}^{<E>}$
5: $(\mathbf{A}^{<S>})^{(N)} \leftarrow \mathbf{Q}^{<S>}$, $\mathcal{G}^{<S>} \leftarrow \mathcal{G}^{<S>} \times_N \mathbf{R}^{<S>}$, $(\mathbf{A}^{<E>})^{(N)} \leftarrow \mathbf{Q}^{<E>}$, and $\mathcal{G}^{<E>} \leftarrow \mathcal{G}^{<E>} \times_N \mathbf{R}^{<E>}$
6: **repeat**
7: **for** $k = 1 \ldots N-1$ **do**
8: update $\tilde{\mathbf{A}}^{(k)}$ by computing Equation (5) and orthogonalizing it with QR decomposition
9: **end for**
10: update $\tilde{\mathbf{A}}^{(N)}$ by computing Equation (7) and orthogonalizing it with QR decomposition
11: update core tensor $\tilde{\mathcal{G}}$ by computing Equation (8)
12: **until** the variation of an error is less than $\epsilon$ or the number of iterations is larger than the maximum number of iterations
13: **return** $\tilde{\mathbf{A}}^{(k)}$ for $k = 1, \ldots, N$ and $\tilde{\mathcal{G}}$

In contrast to naively computing Equation (3) with $\tilde{\mathbf{X}}_{(n)}$, Zoom-Tucker efficiently computes Equation (3) by exploiting preprocessed results obtained in the preprocessing phase.

Before describing an efficient update procedure, we introduce a useful lemma (see the proof in Appendix B.2).

Lemma 2. *Let $\mathcal{S} \in \mathbb{R}^{J \times \cdots \times J}$ and $\mathcal{S}' \in \mathbb{R}^{J \times \cdots \times J}$ be $N$-order tensors, and $\mathbf{U}^{(n)}$ and $\mathbf{V}^{(n)}$ for $n = 1, .., n-1, n+1, .., N$ be matrices of size $I \times J$. Assume our goal is to compute the following equation:*

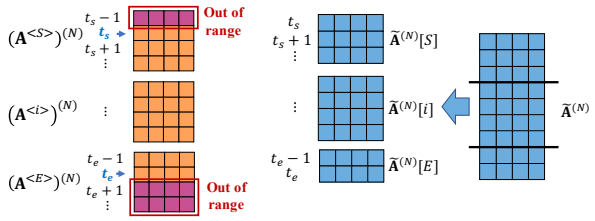
$$\mathbf{S}_{(n)} \left( \otimes_{k \neq n}^{N} \mathbf{U}^{(k)T} \mathbf{V}^{(k)} \right) \mathbf{S'}_{(n)}^{T} \tag{4}$$

*where $\mathbf{S}_{(n)}$ and $\mathbf{S'}_{(n)}$ are the mode-n matricized version of $\mathcal{S}$ and $\mathcal{S}'$, respectively. Naively computing Equation (4) by first computing $\otimes_{k \neq n}^{N} \mathbf{U}^{(k)T} \mathbf{V}^{(k)}$ and multiply with the remaining matrices requires $\mathcal{O}(NIJ^2 + J^{2N} + J^{N+1})$ time and $\mathcal{O}(J^{2N} + NIJ)$ space. Instead, exploiting Equation (1) enables to compute Equation (4) efficiently: $\mathcal{O}(NIJ^2 + NJ^{N+1})$ time and $\mathcal{O}(J^N + NIJ)$ space.* □

For all $n = 1, .., N$, $\mathbf{C}^{(n)}$ is computed based on Lemma 2, by replacing $\mathbf{S}_{(n)}$, $\mathbf{U}^{(k)}$, $\mathbf{V}^{(k)}$, and $\mathbf{S'}_{(n)}$ with $\tilde{\mathbf{G}}_{(n)}$, $\tilde{\mathbf{A}}^{(k)}$, $\tilde{\mathbf{A}}^{(k)}$, and $\tilde{\mathbf{G}}_{(n)}$, respectively.

*3.2.2 Adjusting edge blocks of time range query (Step S2).* Before updates, we adjust the Tucker results of $\mathcal{X}^{<S>}$ and $\mathcal{X}^{<E>}$, the temporal block tensors corresponding to $t_s$ and $t_e$ of the given time range $[t_s, t_e]$, respectively. The temporal factor matrices $(\mathbf{A}^{<S>})^{(N)}$ of $\mathcal{X}^{<S>}$ and $(\mathbf{A}^{<E>})^{(N)}$ of $\mathcal{X}^{<E>}$ may contain the rows that are not included in the range (see Figure 5(a)). To fit to the given time range, we need to remove the non-included rows of $(\mathbf{A}^{<S>})^{(N)}$ and $(\mathbf{A}^{<E>})^{(N)}$, and adjust the Tucker results of $\mathcal{X}^{<S>}$ and $\mathcal{X}^{<E>}$.

Let $p$ be $S$ or $E$. For the temporal factor matrix $(\mathbf{A}^{<p>})^{(N)}$ of $\mathcal{X}^{<p>}$ in the range, Zoom-Tucker obtains the manipulated temporal factor matrix $(\bar{\mathbf{A}}^{<p>})^{(N)}$ by removing the rows of $(\mathbf{A}^{<p>})^{(N)}$ that are not included in the time range (line 3 in Algorithm 2). Next, we perform QR decomposition to make $(\bar{\mathbf{A}}^{<p>})^{(N)}$ maintain




(a) Example of adjustment (b) Example of division

**Figure 5: Examples of adjustment (Section 3.2.2) and division (Section 3.2.3).**

column-orthogonality (line 4 in Algorithm 2); we use $(\mathbf{Q}^{<p>})^{(N)}$ as the temporal factor matrix of $\mathcal{X}^{<p>}$ and update the core tensor $\mathcal{G}^{<p>} \leftarrow \mathcal{G}^{<p>} \times_N (\mathbf{R}^{<p>})^{(N)}$ where $(\mathbf{Q}^{<p>})^{(N)}$ and $(\mathbf{R}^{<p>})^{(N)}$ are the results of QR decomposition (line 5 in Algorithm 2).

*3.2.3 Efficient update of factor matrices (Step S3).* We present how to efficiently update the factor matrix of the non-temporal modes and the temporal mode.

**Updating factor matrix of non-temporal modes.** Consider updating the $n$-th factor matrix, which corresponds to a non-temporal mode. A naive approach is to reconstruct $\tilde{\mathbf{X}}_{(n)}$ from the Tucker results of the preprocessing phase and compute Equation (3). However, it requires large time and space costs since the reconstructed tensor is much larger than the preprocessed results. Our main ideas are to 1) elaborately decouple $\tilde{\mathbf{X}}_{(n)} \left( \otimes_{k \neq n}^{N} \tilde{\mathbf{A}}^{(k)} \right)$ block by block using the preprocessed results, and 2) carefully determine the order of computations, which significantly reduces time and space costs compared to the naive approach. We derive Equation (5) in Lemma 3 to update $\tilde{\mathbf{A}}^{(n)}$ (see the proof in Appendix B.3).

Lemma 3 (Updating factor matrix of a non-temporal mode). *Assume that $\tilde{\mathbf{X}}_{(n)}$ is replaced with the preprocessed results (i.e., $(\mathbf{A}^{<i>})^{(n)}$ and $\mathcal{G}^{<i>}$). Then, the following equation is equal to Equation (3) in Lemma 1 for n-th mode:*

$$\tilde{\mathbf{A}}^{(n)} \leftarrow \sum_{i=S}^{E} (\mathbf{A}^{<i>})^{(n)} (\mathbf{B}^{<i>})^{(n)} \left( \mathbf{C}^{(n)} \right)^{-1} \tag{5}$$

*where the i-th block matrix $(\mathbf{B}^{<i>})^{(n)}$ of the n-th mode is*

$$(\mathbf{B}^{<i>})^{(n)} = \mathbf{G}_{(n)}^{<i>} \left( (\mathbf{A}^{<i>})^{(N)T} \tilde{\mathbf{A}}^{(N)}[i] \otimes \left( \otimes_{k \neq n}^{N-1} (\mathbf{A}^{<i>})^{(k)T} \tilde{\mathbf{A}}^{(k)} \right) \right) \tilde{\mathbf{G}}_{(n)}^{T}, \tag{6}$$

*and $\mathbf{C}^{(n)}$ is defined in Lemma 1. $(\mathbf{A}^{<i>})^{(k)}$ is the k-th factor matrix of the temporal block tensor $\mathcal{X}^{<i>}$, and $\mathbf{G}_{(n)}^{<i>}$ is the mode-n matricized version of the core tensor of $\mathcal{X}^{<i>}$. $\tilde{\mathbf{A}}^{(N)}[i]$ is a sub-matrix of the temporal factor matrix $\tilde{\mathbf{A}}^{(N)}$ such that,*

$$\begin{bmatrix} \tilde{\mathbf{A}}^{(N)}[S] \\ \vdots \\ \tilde{\mathbf{A}}^{(N)}[E] \end{bmatrix} = \tilde{\mathbf{A}}^{(N)}$$

*To compute $(\mathbf{A}^{<i>})^{(N)T} \tilde{\mathbf{A}}^{(N)}[i]$, we split $\tilde{\mathbf{A}}^{(N)}$ into sub-matrices $\tilde{\mathbf{A}}^{(N)}[i]$ ($i = S, .., E$) along the time dimension (see Figure 5(b)); the size of $\tilde{\mathbf{A}}^{(N)}[i]$ for $i = S+1, .., E-1$ is $b \times J_N$, and that of $\tilde{\mathbf{A}}^{(N)}[S]$ and $\tilde{\mathbf{A}}^{(N)}[E]$ is $(b - r_S) \times J_N$ and $(b - r_E) \times J_N$, respectively, where $r_S$ and $r_E$ are the number of the rows removed with respect to $t_s$ and $t_e$, respectively.* □

Zoom-Tucker efficiently updates $\tilde{\mathbf{A}}^{(n)}$ with Equation (5). Zoom-Tucker minimizes the intermediate data and reduces the high computational cost by independently computing $\mathbf{C}^{(n)}$ and $(\mathbf{B}^{<i>})^{(n)}$ for $i = S, ..., E$. Note that $(\mathbf{B}^{<i>})^{(n)}$ for $i = S, ..., E$ is computed based on Lemma 2, by replacing $\mathbf{S}_{(n)}$, $\mathbf{U}^{(k)}$, $\mathbf{V}^{(k)}$, and $\mathbf{S'}_{(n)}$ with $\mathbf{G}^{<i>}_{(n)}$, $(\mathbf{A}^{<i>})^{(k)}$, $\tilde{\mathbf{A}}^{(k)}$ (or $\tilde{\mathbf{A}}^{(N)}[i]$), and $\tilde{\mathbf{G}}_{(n)}$, respectively. Next, we obtain $\tilde{\mathbf{A}}^{(n)}$ by summing up the results of $(\mathbf{A}^{<i>})^{(n)}(\mathbf{B}^{<i>})^{(n)}\left(\mathbf{C}^{(n)}\right)^{-1}$ for $i = S, ..., E$. For orthogonalization, we then update $\tilde{\mathbf{A}}^{(n)} \leftarrow \tilde{\mathbf{Q}}^{(n)}$ after QR decomposition $\tilde{\mathbf{A}}^{(n)} \rightarrow \tilde{\mathbf{Q}}^{(n)}\tilde{\mathbf{R}}^{(n)}$ (line 8 in Algorithm 2).

**Updating factor matrix of temporal mode.** The goal is to update the factor matrix $\tilde{\mathbf{A}}^{(N)}$ of the temporal mode by using the preprocessed results instead of $\tilde{\mathbf{X}}_{(N)}$. Reconstructing $\tilde{\mathbf{X}}_{(N)}$ requires high space and time costs in Equation (3). Based on our ideas used for the non-temporal modes, we efficiently update $\tilde{\mathbf{A}}^{(N)}$ by computing Equation (7) in Lemma 4 (see the proof in Appendix B.4).

Lemma 4 (Updating factor matrix of temporal mode). *Assume that $\tilde{\mathbf{X}}_{(N)}$ is replaced with the preprocessed results (i.e., $(\mathbf{A}^{<i>})^{(n)}$ and $\mathcal{G}^{<i>}$). Then, the following equation is equal to Equation* (3) *in Lemma 1 for the temporal mode:*

$$\tilde{\mathbf{A}}^{(N)} \leftarrow \begin{bmatrix} (\mathbf{A}^{<S>})^{(N)}(\mathbf{B}^{<S>})^{(N)} \\ \vdots \\ (\mathbf{A}^{<E>})^{(N)}(\mathbf{B}^{<E>})^{(N)} \end{bmatrix} \left(\mathbf{C}^{(N)}\right)^{-1} \tag{7}$$

*where the $i$-th matrix $(\mathbf{B}^{<i>})^{(N)} \in \mathbb{R}^{J_N \times J_N}$ for $i = S, ..., E$ is*

$$(\mathbf{B}^{<i>})^{(N)} = \mathbf{G}^{<i>}_{(N)} \left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<i>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}^{T}_{(N)}$$

*$(\mathbf{A}^{<i>})^{(k)}$ is the $k$-th factor matrix of $\mathcal{X}^{<i>}$, $\mathbf{G}^{<i>}_{(N)}$ is the mode-$N$ matricized version of the core tensor of $\mathcal{X}^{<i>}$, and $\mathbf{C}^{(N)}$ is equal to $\tilde{\mathbf{G}}_{(N)}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}^{T}_{(N)}$.* □

We obtain $\tilde{\mathbf{A}}^{(N)}$ by using $(\mathbf{C}^{(N)})^{-1}$, $(\mathbf{A}^{<i>})^{(N)}$, and $(\mathbf{B}^{<i>})^{(N)}$ for $i = S, ..., E$. Zoom-Tucker efficiently updates $\tilde{\mathbf{A}}^{(N)}$ by independently computing $\mathbf{C}^{(N)}$ and $(\mathbf{B}^{<i>})^{(N)}$ for $i = S, ..., E$. $(\mathbf{B}^{<i>})^{(N)}$ is efficiently computed based on Lemma 2, by replacing $\mathbf{S}_{(n)}$, $\mathbf{U}^{(k)}$, $\mathbf{V}^{(k)}$, and $\mathbf{S'}_{(n)}$ with $\mathbf{G}^{<i>}_{(N)}$, $(\mathbf{A}^{<i>})^{(k)}$, $\tilde{\mathbf{A}}^{(k)}$, and $\tilde{\mathbf{G}}_{(N)}$, respectively. For orthogonalization, we update $\tilde{\mathbf{A}}^{(N)} \leftarrow \tilde{\mathbf{Q}}^{(N)}$ after QR decomposition $\tilde{\mathbf{A}}^{(N)} \rightarrow \tilde{\mathbf{Q}}^{(N)}\tilde{\mathbf{R}}^{(N)}$ (line 10 in Algorithm 2).

*3.2.4 Efficient update of core tensor (Step S4).* At the end of each iteration, Zoom-Tucker updates the core tensor using the factor matrices: $\tilde{\mathbf{G}}_{(N)} \leftarrow \tilde{\mathbf{A}}^{(N)T}\tilde{\mathbf{X}}_{(N)}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)$ (mode-$N$ matricization of line 8 in Algorithm 3). We efficiently compute the core tensor by avoiding reconstruction of $\tilde{\mathbf{X}}_{(N)}$ and carefully determining the order of computation. We replace $\tilde{\mathbf{X}}_{(N)}$ with the preprocessed results and refine the equation with block decoupling and the *mixed-product property* (see Equation (10) in Appendix B.4).

$$\tilde{\mathbf{G}}_{(N)} \leftarrow \left(\sum_{i=S}^{E}(\tilde{\mathbf{A}}^{(N)T}[i])(\mathbf{A}^{<i>})^{(N)}\mathbf{G}^{<i>}_{(N)}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<i>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\right) \tag{8}$$

With Equation (8), Zoom-Tucker efficiently updates $\tilde{\mathcal{G}}$, reducing the intermediate data and the computational cost. For each $i$, Zoom-Tucker computes $(\tilde{\mathbf{A}}^{(N)T}[i])(\mathbf{A}^{<i>})^{(N)}\mathbf{G}^{<i>}_{(N)}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<i>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)$ after transforming it into $n$-mode products as in Equation (1). After that, Zoom-Tucker obtains $\tilde{\mathbf{G}}_{(N)}$ by summing up the results and reshape it to the core tensor $\tilde{\mathcal{G}}$ (line 11 in Algorithm 2).

**Table 2: Time and space complexities of Zoom-Tucker and other methods for a time range $[t_s, t_e]$. The optimal complexities are in bold. $I$, $J$, $M$, $N$, and $l_{[t_s,t_e]}$ are described in Section 3.3. $S$ is a sampling rate for MACH.**

| Algorithm | Time | Space |
|---|---|---|
| Zoom-Tucker | $\mathbf{O(l_{[t_s,t_e]}IMN^2J^2/b)}$ | $\mathbf{O(l_{[t_s,t_e]}NIJ/b)}$ |
| D-Tucker [12] | $O(l_{[t_s,t_e]}I^{N-2}MNJ^2)$ | $O(l_{[t_s,t_e]}I^{N-2}J)$ |
| Tucker-ALS | $O(l_{[t_s,t_e]}I^{N-1}MNJ)$ | $O(l_{[t_s,t_e]}I^{N-1})$ |
| MACH [35] | $O(Sl_{[t_s,t_e]}I^{N-1}MNJ)$ | $O(Sl_{[t_s,t_e]}I^{N-1})$ |
| RTD [5] | $O(l_{[t_s,t_e]}I^{N-1}MN)$ | $O(l_{[t_s,t_e]}I^{N-1})$ |
| Tucker-ts [25] | $O(l_{[t_s,t_e]}I^{N-1}N + MNIJ^N)$ | $O(l_{[t_s,t_e]}I^{N-1} + NIJ^N)$ |
| Tucker-ttmts [25] | $O(l_{[t_s,t_e]}I^{N-1}N + MNIJ^{2N-2})$ | $O(l_{[t_s,t_e]}I^{N-1} + NIJ^N)$ |

## 3.3 Analysis

We analyze the time and space complexities of Zoom-Tucker in the preprocessing phase and the query phase. We assume that $I = I_1 = ... = I_{N-1}$, and $J = J_1 = ... = J_N$. $M$ is the number of iterations, $l_{[t_s,t_e]} = t_e - t_s + 1$ is the length of a time range query, $N$ is the order of a given tensor, $I$ is the dimensionality, $b$ is the block size, $B$ is the number of blocks, and $J$ is the rank. All proofs are summarized in Appendices B.5 to B.8.

**Time complexity.** We analyze computational cost of Zoom-Tucker in the preprocessing phase and the query phase.

Theorem 1. *The preprocessing phase takes $O(MNI^{N-1}JbB)$ time.*

Theorem 2. *Given a time range query $[t_s, t_e]$, the query phase of Zoom-Tucker takes $\mathbf{O}\left(MNJ^2 l_{[t_s,t_e]}\left(1 + \frac{NI}{b} + \frac{NJ^{N-1}}{b}\right)\right)$ time.* □

**Space complexity.** We provide analysis for the space cost of Zoom-Tucker in the preprocessing phase and the query phase.

Theorem 3. *Zoom-Tucker requires $\mathbf{O}\left(NIJ(\lceil\frac{I_N}{b}\rceil) + I_N J\right)$ space to store the Tucker results in the preprocessing phase.* □

Theorem 4. *Given a time range query $[t_s, t_e]$, Zoom-Tucker requires $\mathbf{O}\left(NIJ(\lceil\frac{l_{[t_s,t_e]}}{b}\rceil) + Jl_{[t_s,t_e]}\right)$ space in the query phase.* □

Table 2 shows the time and space complexities of Zoom-Tucker and competitors for a given time range query $[t_s, t_e]$. The time and space complexities of Zoom-Tucker mainly depend on $I$ and $l_{[t_s,t_e]}$. We also note that the block size $b$ reduces the complexities of Zoom-Tucker. We compare the time and space complexities of Zoom-Tucker with those of the second-best method, D-Tucker. For both time and space complexities, the result of dividing the complexity of Zoom-Tucker by that of D-Tucker is $\frac{N}{I^{N-3}b}$. Zoom-Tucker has better time and space complexities than D-Tucker since $I^{N-3}b$ is larger than $N$ in real-world datasets; for example, in the experiments, we use 50 as the default block size $b$ while the order of the real-world datasets is 3 or 4. As $b$ increases, the space complexity of the preprocessing and the query phases, and the time complexity of the query phase decrease; however, a large block size $b$ can provoke a high reconstruction error for a narrow time range query since the preprocessing phase with the large $b$ cannot capture local information. In Section 4.4, we experimentally find a block size that enables the preprocessing phase to capture local information with low reconstruction errors for narrow time range queries.

**Table 3: Description of real-world tensor datasets.**

| Dataset | Dimensionality | Length $l_{[t_s,t_e]}$ of Time Range | Summary |
|---|---|---|---|
| Boats[1] [37] | 320 × 240 × 7000 | (128, 2048) | Video |
| Walking Video [25] | 1080 × 1980 × 2400 | (128, 2048) | Video |
| Stock[3] | 3028 × 54 × 3050 | (128, 2048) | Time series |
| Traffic[4] [30] | 1084 × 96 × 2000 | (64, 1024) | Traffic volume |
| FMA[5] [8] | 7994 × 1025 × 700 | (32, 512) | Music |
| Absorb[6] | 192 × 288 × 30 × 1200 | (64, 1024) | Climate |

# 4 EXPERIMENT

We present experimental results to answer the following questions.

- Q1 **Performance Trade-off (Section 4.2).** Does Zoom-Tucker provide the best trade-off between query time and reconstruction error?
- Q2 **Space Cost (Section 4.3).** What is the space cost of Zoom-Tucker and competitors for preprocessed results?
- Q3 **Effects of the block size $b$ (Section 4.4).** How does a block size $b$ affect query time and reconstruction error of Zoom-Tucker?
- Q4 **Discovery (Section 4.5).** What pattern does Zoom-Tucker discover in different time ranges?

## 4.1 Experimental Settings

**Machine.** We run experiments on a workstation with a single CPU (Intel Xeon E5-2630 v4 @ 2.2GHz), and 512GB memory.

**Dataset.** We use six real-world dense tensors in Table 3. Boats[1] [37] and Walking Video[2] [25] datasets contain grayscale videos in the form of (height, width, time; value). Stock dataset[3] contains 5 basic features (open price, high price, low price, close price, trade volume) and 49 technical indicators features of Korea Stocks. Stock dataset has the form of (stock, features, date; value). The basic features are collected daily from *Jan. 2, 2008* to *May 6, 2020*. Traffic dataset[4] [30] contains traffic volume information in the form of (sensor, frequency, time; measurement). FMA dataset[5] [8] contains music information: (song, frequency, time; value). We convert a time series into an image of a log-power spectrogram for each song. Absorb dataset[6] is about absorption of aerosol in the form of (longitudes, latitudes, altitude, time; measurement).

**Competitors.** We compare Zoom-Tucker with 6 Tucker decomposition methods based on ALS approach. Zoom-Tucker and other methods are implemented in MATLAB (R2019b). We use the open sourced codes for 4 competitors: D-Tucker[7], Tucker-ALS [3], Tucker-ts[8], and Tucker-ttmts[8]. For MACH, we run Tucker-ALS in Tensor Toolbox [3] for a sampled tensor after sampling elements of a tensor; we use our implementation for a sampling scheme. We use the source code of RTD [5] provided by the authors.

**Parameters.** The parameter settings used for experiments are described in Appendix C.

**Implementation details.** In the time range query problem, Zoom-Tucker, D-Tucker, and MACH preprocess a given tensor, and then perform Tucker decomposition for a time range query using preprocessed results included in the range. In contrast, Tucker-ALS and RTD perform Tucker decomposition using a sub-tensor included in a time range query. Although Tucker-ts and Tucker-ttmts have a preprocessing phase, they also perform Tucker decomposition from scratch for a time range query since there is an inseparable preprocessed result along the time dimension.

[1]http://changedetection.net/
[2]https://github.com/OsmanMalik/tucker-tensorsketch
[3]https://datalab.snu.ac.kr/zoomtucker
[4]https://github.com/florinsch/BigTrafficData
[5]https://github.com/mdeff/fma
[6]https://www.earthsystemgrid.org/
[7]https://datalab.snu.ac.kr/dtucker/
[8]https://github.com/OsmanMalik/tucker-tensorsketch

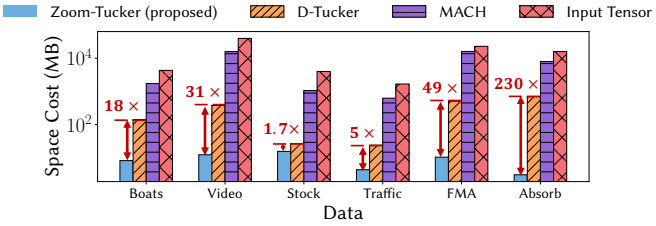



**Figure 6: Space cost for storing preprocessed results. *Input Tensor* corresponds to the space cost of Tucker-ALS, Tucker-ts, Tucker-ttmts, and RTD. Zoom-Tucker requires up to 230× less space than competitors.**

**Reconstruction error.** Given an input tensor $\mathcal{X}$ and the reconstruction $\hat{\mathcal{X}}$ from the output of Tucker decomposition, reconstruction error is defined as $\frac{\|\mathcal{X}-\hat{\mathcal{X}}\|_F^2}{\|\mathcal{X}\|_F^2}$. Reconstruction error describes how well the reconstruction $\hat{\mathcal{X}}$ of Tucker decomposition represents an input tensor $\mathcal{X}$.

## 4.2 Trade-off between Query Time and Reconstruction Error (Q1)

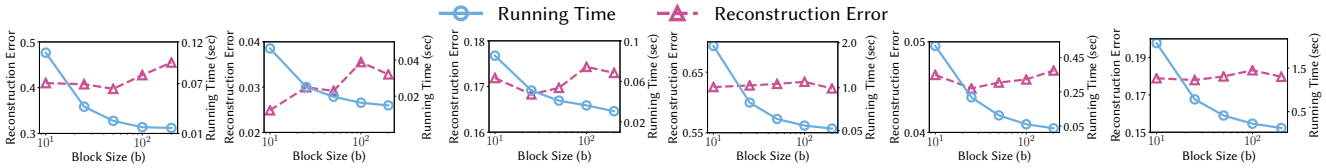
We compare the running time and reconstruction error of Zoom-Tucker with those of competitors for various time ranges. For each dataset, we use the narrowest and the widest time ranges among the ranges described in Table 3. Figure 2 shows that Zoom-Tucker is the closest method to the best point with the smallest error and running time. Zoom-Tucker is up to 171.9× and 111.9× faster than the second-fastest method, in narrow and wide time ranges, respectively, with similar errors.

## 4.3 Space Cost (Q2)

We compare the storage cost of Zoom-Tucker with those of competitors for storing preprocessed results. Note that memory requirements for a time range query are proportional to the storage cost since preprocessed results or an input tensor is the dominant term in the space cost. Figure 6 shows that Zoom-Tucker requires the lowest space; Zoom-Tucker requires up to 230× less space than the second-best method D-Tucker. Zoom-Tucker has more compression rate on the 4-order tensor, Absorb dataset.

## 4.4 Effects of Block Size $b$ (Q3)

We investigate the effects of block size $b$ on running time and reconstruction error of Zoom-Tucker. We use block sizes 10, 25, 50, 100, and 200 on Stock, Traffic, and Absorb datasets. As shown in Figures 7(a) to 7(c), there are trade-off relationships between running time and reconstruction error for narrow time range queries. In Figures 7(d) to 7(f), the running time of Zoom-Tucker is inversely proportional to $b$ for a wide range query while the reconstruction error is not sensitive to $b$. A large $b$ prevents the preprocessing phase from capturing local information so that it is challenging to serve narrow time range queries. For wide time range queries, local information has little effect on reconstruction errors since




(a) Stock data (128) (b) Traffic data (64) (c) Absorb data (64) (d) Stock data (2048) (e) Traffic data (1024) (f) Absorb data (1024)

**Figure 7: Sensitivity with respect to block size $b$ on Stock, Traffic, and Absorb datasets. Numbers after the data name represent the length of time ranges; e.g., (128) means the length of time range is $t_e - t_s + 1 = 128$ timesteps. (a,b,c) There are trade-off relationships between running time and reconstruction error for narrow time range queries. (d,e,f) For wide time range queries, the running times decrease while the errors do not change much, as block size increases.**

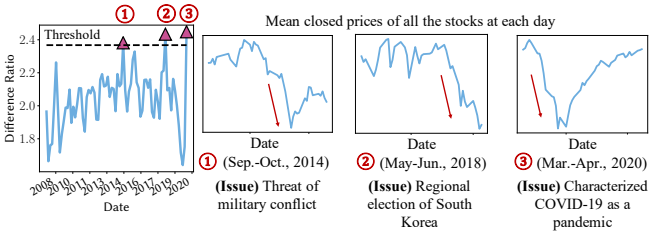



**Figure 8: Anomalous two-month ranges and their related events, found by Zoom-Tucker.**

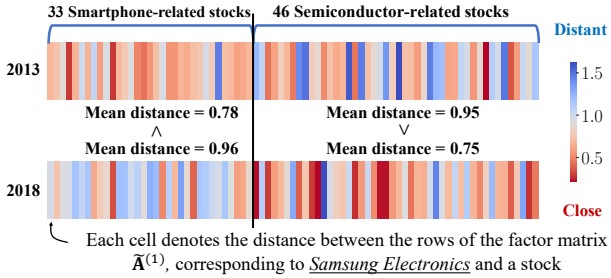



**Figure 9: Cosine distance between feature vectors of *Samsung Electronics* and other stocks related to smartphone or semiconductors in 2013 and 2018. Zoom-Tucker helps capture the clear change of the trend, where *Samsung Electronics* is more close to smartphone-related stocks in 2013, but to semiconductor-related stocks in 2018.**

capturing widespread patterns is more beneficial in reducing errors. Therefore, we select 50, which is the largest value providing small errors for narrow time range queries, for the default block size to preprocess all datasets in other experimental sections.

## 4.5 Discovery (Q4)

On Stock dataset, we discover interesting results by answering various time range queries with Zoom-Tucker.

**Finding anomalous ranges.** The goal is to find narrow time ranges that are anomalous, compared to the entire time range. For the goal, we select every consecutive two-month interval from Jan. 1, 2008 to Apr. 30, 2020, perform Tucker decomposition for each of the intervals using Zoom-Tucker, and find anomalous ranges that deviate the most from the entire ranges. Given a two-month range $r$, and its corresponding sub-tensor $\tilde{\mathcal{X}}$, we compute the anomaly score for $r$ using the difference ratio $\frac{\|\tilde{\mathcal{X}}-\hat{\mathcal{Y}}\|_F^2}{\|\tilde{\mathcal{X}}-\hat{\mathcal{Z}}\|_F^2}$ where $\hat{\mathcal{Y}}$ and $\hat{\mathcal{Z}}$ are the sub-tensors for $r$ reconstructed from the Tucker results of 1) the entire range query, and 2) the two-month range query, respectively.

The leftmost plot of Figure 8 shows the difference ratios and the top three anomalous ranges where the threshold indicates 2 standard deviations from the mean. The right three plots of Figure 8 show that the three anomalies follow the similar plunging pattern of prices from issues affecting the stock market.

**Analyzing trend change.** We analyze the change of yearly trend of *Samsung Electronics* in the years 2013 and 2018. For each of the range (year 2013 or 2018), we perform Zoom-Tucker and get the feature matrix $\tilde{\mathbf{A}}^{(1)}$ each of whose rows contain the latent features of a stock. We also manually pick 33 smartphone-related stocks and 46 semiconductor-related stocks, and compare the cosine distance between the latent feature vectors of each stock and *Samsung Electronics*.

Figure 9 shows the result. Note that there is a clear change of the distances between year 2013 and 2018: *Samsung Electronics* is more close to smartphone-related stocks in 2013, but to semiconductor-related stocks in 2018. This result exactly reflects the sales trend of *Samsung Electronics*; the annual sales of its smartphone division are 3.7× larger than those of its semiconductor division in 2013, while in 2018 the annual sales of its semiconductor division are 30% larger than those of its smartphone division. Zoom-Tucker enables us to quickly and accurately capture this trend change.

## 5 RELATED WORK

We review related works for efficient tensor decomposition, block-based tensor decomposition, and time range query for tensors.

**Efficient tensor decomposition.** Many works have been devoted to computing efficient tensor decomposition in various settings. Previous works [2, 14, 15, 28, 38] develop efficient tensor decomposition methods on distributed systems. Several tensor decomposition methods [21, 26, 27, 31, 32, 34] have been proposed for sparse tensors; however, they target performing tensor decomposition only once for the whole data. There are several works [7, 9, 18, 22, 33, 39] that perform tensor decomposition in streaming settings. Zoom-Tucker is different from the above methods since it handles arbitrary time range queries in a single machine.

**Tensor decomposition with block-wise computation.** Many tensor decomposition methods have exploited block-wise computation for parallel computation. [6, 23, 29] proposed parallel CP decomposition methods which perform CP decomposition block by block and then concatenate the results of the blocks. Austin et al. [2] proposed a distributed algorithm that computes $n$-mode product, gram matrix, and eigenvectors with the small blocks of a given tensor. Unlike the above methods which do not consider

time ranges, the goal of our Zoom-Tucker is to quickly provide Tucker decomposition results for a given time range query.

**Time range query for tensors.** Zoom-SVD [11] deals with the time range query problem, but it is suitable only for multiple time series data represented as a matrix. Although there is no existing method that precisely addresses the time range query problem for tensors, there are several methods [12, 25, 35] that can be adapted to solve the problem. They perform a preprocessing phase by exploiting a sampling technique [35] or randomized SVD [12] before the query phase, and then obtain Tucker results using the preprocessed results in the query phase. However, they do not satisfy the desired properties for the solution: fast running time, low space cost, and accuracy. On the other hand, Zoom-Tucker efficiently and accurately provides answers to time range queries by exploiting the preprocessed results.

## 6 CONCLUSIONS

In this work, we propose Zoom-Tucker, an efficient Tucker decomposition method to discover latent factors in a given time range from a temporal tensor. Zoom-Tucker efficiently answers diverse time range queries with the preprocessing phase and the query phase. In the preprocessing phase, Zoom-Tucker lays the groundwork for an efficient time range query by compressing sub-tensors along time dimension block by block. Given a time range query in the query phase, Zoom-Tucker elaborately stitches compressed results reducing computational cost and space cost. Experiments show that Zoom-Tucker is up to 171.9× faster and requires up to 230× less space than existing methods, with comparable accuracy to competitors. With Zoom-Tucker, we discover interesting patterns including anomalous ranges and trend changes in a real-world stock dataset. Future research includes extending the method for sparse tensors.

## ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) funded by MSIT(2019R1A2C2004990). The Institute of Engineering Research and ICT at Seoul National University provided research facilities for this work. U Kang is the corresponding author.

## REFERENCES

[1] Dawon Ahn, Sangjun Son, and U Kang. 2020. Gtensor: Fast and Accurate Tensor Analysis System using GPUs. In *CIKM*. ACM, 3361–3364.
[2] Woody Austin, Grey Ballard, and Tamara G. Kolda. 2016. Parallel Tensor Compression for Large-Scale Scientific Data. In *IPDPS*. IEEE Computer Society, 912–922.
[3] Brett W. Bader, Tamara G. Kolda, et al. 2017. MATLAB Tensor Toolbox Version 3.0-dev. Available online. https://www.tensortoolbox.org
[4] Xiaochun Cao, Xingxing Wei, Yahong Han, and Dongdai Lin. 2015. Robust Face Clustering Via Tensor Decomposition. *IEEE Trans. Cybernetics* 45, 11 (2015), 2546–2557.
[5] Maolin Che and Yimin Wei. 2019. Randomized algorithms for the approximations of Tucker and the tensor train decompositions. *Adv. Comput. Math.* 45, 1 (2019), 395–428.
[6] Dan Chen, Yangyang Hu, Lizhe Wang, Albert Y. Zomaya, and Xiaoli Li. 2017. H-PARAFAC: Hierarchical Parallel Factor Analysis of Multidimensional Big Data. *TPDS* 28, 4 (2017), 1091–1104.
[7] Dongjin Choi, Jun-Gi Jang, and U Kang. 2019. S3CMTF: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization. *PLOS ONE* 14, 6 (06 2019), 1–20.
[8] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. 2017. FMA: A Dataset for Music Analysis. In *ISMIR*. arXiv:1612.01840 https://arxiv.org/abs/1612.01840
[9] Ekta Gujral, Ravdeep Pasricha, and Evangelos E. Papalexakis. 2018. SamBaTen: Sampling-based Batch Incremental Tensor Decomposition. In *SDM*. 387–395.
[10] Heng Huang, Chris H. Q. Ding, Dijun Luo, and Tao Li. 2008. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *SIGKDD*. ACM, 327–335.
[11] Jun-Gi Jang, Dongjin Choi, Jinhong Jung, and U Kang. 2018. Zoom-SVD: Fast and Memory Efficient Method for Extracting Key Patterns in an Arbitrary Time Range. In *CIKM*. ACM, 1083–1092.
[12] Jun-Gi Jang and U Kang. 2020. D-Tucker: Fast and Memory-Efficient Tucker Decomposition for Dense Tensors. In *ICDE*. IEEE, 1850–1853.
[13] Byungsoo Jeon, Inah Jeon, Lee Sael, and U Kang. 2016. SCouT: Scalable coupled matrix-tensor factorization - algorithm and discoveries. In *ICDE*. IEEE Computer Society, 811–822.
[14] Inah Jeon, Evangelos E. Papalexakis, U. Kang, and Christos Faloutsos. 2015. HaTen2: Billion-scale tensor decompositions. In *ICDE*. 1047–1058.
[15] Oguz Kaya and Bora Uçar. 2015. Scalable sparse tensor decompositions in distributed memory systems. In *SC*. ACM, 77:1–77:11.
[16] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. *CoRR* abs/1511.06530 (2015). arXiv:1511.06530 http://arxiv.org/abs/1511.06530
[17] Tamara G. Kolda and Brett W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (2009), 455–500.
[18] Taehyung Kwon, Inkyu Park, Dongjin Lee, and Kijung Shin. 2021. SliceNStitch: Continuous CP Decomposition of Sparse Tensor Streams. *CoRR* abs/2102.11517 (2021).
[19] Timothée Lacroix, Guillaume Obozinski, and Nicolas Usunier. 2020. Tensor Decompositions for Temporal Knowledge Base Completion. In *ICLR*. OpenReview.net.
[20] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. On the Best Rank-1 and Rank-($R_1$ , $R_2$, ... , $R_N$) Approximation of Higher-Order Tensors. *SIAM J. Matrix Analysis Applications* 21, 4 (2000), 1324–1342.
[21] Dongha Lee, Jaehyung Lee, and Hwanjo Yu. 2018. Fast Tucker Factorization for Large-Scale Tensor Completion. In *ICDM*. IEEE Computer Society, 1098–1103.
[22] Dongjin Lee and Kijung Shin. 2021. Robust Factorization of Real-world Tensor Streams with Patterns, Missing Values, and Outliers. *CoRR* abs/2102.08466 (2021).
[23] Xinsheng Li, Shengyu Huang, K. Selçuk Candan, and Maria Luisa Sapino. 2016. 2PCP: Two-phase CP decomposition for billion-scale dense tensors. In *ICDE*. IEEE Computer Society, 835–846.
[24] Yu Liu, Quanming Yao, and Yong Li. 2020. Generalizing Tensor Decomposition for N-ary Relational Knowledge Bases. In *WWW*. ACM / IW3C2, 1104–1114.
[25] Osman Asif Malik and Stephen Becker. 2018. Low-Rank Tucker Decomposition of Large Tensors Using TensorSketch. In *NeurIPS*. 10117–10127.
[26] Sejoon Oh, Namyong Park, Jun-Gi Jang, Lee Sael, and U Kang. 2019. High-Performance Tucker Factorization on Heterogeneous Platforms. *IEEE Trans. Parallel Distributed Syst.* 30, 10 (2019), 2237–2248.
[27] Sejoon Oh, Namyong Park, Lee Sael, and U. Kang. 2018. Scalable Tucker Factorization for Sparse Tensors - Algorithms and Discoveries. In *ICDE*. 1120–1131.
[28] Namyong Park, Byungsoo Jeon, Jungwoo Lee, and U Kang. 2016. BIGtensor: Mining Billion-Scale Tensor Made Easy. In *CIKM*. ACM, 2457–2460.
[29] Anh Huy Phan and Andrzej Cichocki. 2011. PARAFAC algorithms for large-scale problems. *Neurocomputing* 74, 11 (2011), 1970–1984.
[30] Florin Schimbinschi, Xuan Vinh Nguyen, James Bailey, Chris Leckie, Hai Vu, and Rao Kotagiri. 2015. Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *Big Data*. IEEE, 1019–1024.
[31] Kijung Shin and U Kang. 2014. Distributed Methods for High-Dimensional and Large-Scale Tensor Factorization. In *ICDM*. IEEE Computer Society, 989–994.
[32] Kijung Shin, Lee Sael, and U Kang. 2017. Fully Scalable Methods for Distributed Tensor Factorization. *IEEE Trans. Knowl. Data Eng.* 29, 1 (2017), 100–113.
[33] Shaden Smith, Kejun Huang, Nicholas D. Sidiropoulos, and George Karypis. 2018. Streaming Tensor Factorization for Infinite Data Sources. In *SDM*. SIAM, 81–89.
[34] Shaden Smith, Niranjay Ravindran, Nicholas D. Sidiropoulos, and George Karypis. 2015. SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication. In *IPDPS*. IEEE Computer Society, 61–70.
[35] Charalampos E. Tsourakakis. 2010. MACH: Fast Randomized Tensor Decompositions. In *SDM*. 689–700.
[36] Hongcheng Wang and Narendra Ahuja. 2008. A Tensor Approximation Approach to Dimensionality Reduction. *Int. J. Comput. Vis.* 76, 3 (2008), 217–229.
[37] Yi Wang, Pierre-Marc Jodoin, Fatih Murat Porikli, Janusz Konrad, Yannick Benezeth, and Prakash Ishwar. 2014. CDnet 2014: An Expanded Change Detection Benchmark Dataset. In *CVPR Workshops*. 393–400.
[38] Fan Yang, Fanhua Shang, Yuzhen Huang, James Cheng, Jinfeng Li, Yunjian Zhao, and Ruihao Zhao. 2017. LFTF: A Framework for Efficient Tensor Analytics at Scale. *Proc. VLDB Endow.* 10, 7 (2017), 745–756.
[39] Shuo Zhou, Xuan Vinh Nguyen, James Bailey, Yunzhe Jia, and Ian Davidson. 2016. Accelerating Online CP Decompositions for Higher Order Tensors. In *SIGKDD*. 1375–1384.

# APPENDIX

## A TUCKER-ALS

**Algorithm 3:** Tucker-ALS (HOOI) [17, 20]

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

1: **initialize:** factor matrices $\mathbf{A}^{(n)}$ $(n = 1, \ldots, N)$
2: **repeat**
3: **for** $n = 1, \ldots, N$ **do**
4: $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)T} \cdots \times_{n-1} \mathbf{A}^{(n-1)T} \times_{n+1} \mathbf{A}^{(n+1)T} \cdots \times_N \mathbf{A}^{(N)T}$
5: $\mathbf{A}^{(n)} \leftarrow J_n$ leading left singular vectors of $\mathbf{Y}_{(n)}$
6: **end for**
7: **until** convergence criterion is met;
8: $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \cdots \times_N \mathbf{A}^{(N)T}$

## B PROOFS

### B.1 Proof of Lemma 1

Proof. After fixing all factor matrices except for the $n$-th factor matrix, the partial derivative of the Equation (2) with respect to the factor matrix $\tilde{\mathbf{A}}^{(n)}$ is as follows:

$$\frac{\partial L_{(n)}}{\partial \tilde{\mathbf{A}}^{(n)}} = -2\tilde{\mathbf{X}}_{(n)}(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)})\tilde{\mathbf{G}}_{(n)}^{T} + 2\tilde{\mathbf{A}}^{(n)}\tilde{\mathbf{G}}_{(n)}\left(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(n)}^{T}$$

We set $\frac{\partial L_{(n)}}{\partial \tilde{\mathbf{A}}^{(n)}}$ to zero, and solve the equation with respect to the factor matrix $\tilde{\mathbf{A}}^{(n)}$:

$$\tilde{\mathbf{A}}^{(n)}\left(\tilde{\mathbf{G}}_{(n)}\left(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(n)}^{T}\right) = \tilde{\mathbf{X}}_{(n)}\left(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(n)}^{T}$$
$$\Leftrightarrow \tilde{\mathbf{A}}^{(n)} = \tilde{\mathbf{X}}_{(n)}\left(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(n)}^{T}\left(\mathbf{C}^{(n)}\right)^{-1}$$

□

### B.2 Proof of Lemma 2

Proof. A naive approach computing Equation (4) is to explicitly compute the entire Kronecker product $\left(\otimes_{k\neq n}^{N}\mathbf{U}^{(k)T}\mathbf{V}^{(k)}\right)$ of the size $J^{N-1} \times J^{N-1}$. We compute matrix multiplication between the preceding result $\mathbf{S}_{(n)}$ and $\mathbf{S'}_{(n)}$. Therefore, the time and space complexities are $\mathcal{O}(NIJ^2 + J^{2N} + J^{N+1})$ and $\mathcal{O}(J^{2N} + NIJ)$, respectively.

We compute Equation (4) using $n$-mode product instead of Kronecker product. Let $\mathbf{Z}_{(n)} = \mathbf{S}_{(n)}\left(\otimes_{k\neq n}^{N}\mathbf{U}^{(k)T}\mathbf{V}^{(k)}\right)$ be equal to $\mathbf{I}^{(n)}\mathbf{S}_{(n)}\left(\otimes_{k\neq n}^{N}\mathbf{U}^{(k)T}\mathbf{V}^{(k)}\right)$ where $\mathbf{I}^{(n)} \in \mathbb{R}^{J\times J}$ is an identity matrix. Then, we transform $\mathbf{Z}$ into Equation (9) using Equation (1).

$$\begin{aligned}\mathcal{Z} = \mathcal{S} \times_1 (\mathbf{U}^{(1)T}\mathbf{V}^{(1)})^T \cdots \times_{n-1} (\mathbf{U}^{(n-1)T}\mathbf{V}^{(n-1)})^T \\ \times_n \mathbf{I}^{(n)} \times_{n+1} (\mathbf{U}^{(n+1)T}\mathbf{V}^{(n+1)})^T \cdots \times_N (\mathbf{U}^{(N)T}\mathbf{V}^{(N)})^T\end{aligned} \tag{9}$$

Based on Equation (9), we compute Equation (4) in the following order: 1) $\mathbf{U}^{(k)T}\mathbf{V}^{(k)}$ for $k = 1, ..., n-1, n+1, ..., N$, 2) $\mathbf{Z}_{(n)}$, and 3) $\mathbf{Z}_{(n)}\mathbf{S'}_{(n)}^{T}$. Therefore, the computational cost is $\mathcal{O}(NIJ^2 + NJ^{N+1})$. In addition, the size of intermediate data is always no larger than $J^N$ so that the space complexity is $\mathcal{O}(J^N + NIJ)$. □

### B.3 Proof of Lemma 3

Proof. From Equation (3), we carefully decouple $\tilde{\mathbf{X}}_{(n)}\left(\otimes_{k\neq n}^{N}\tilde{\mathbf{A}}^{(k)}\right)$ block by block so that we represent the term as a summation of block matrices:

$$\tilde{\mathbf{A}}^{(n)} = \left[\mathbf{X}_{(n)}^{<S>} \cdots \mathbf{X}_{(n)}^{<E>}\right]\left(\begin{bmatrix}\tilde{\mathbf{A}}^{(N)}[S]\\ \vdots \\ \tilde{\mathbf{A}}^{(N)}[E]\end{bmatrix} \otimes \left(\otimes_{k\neq n}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)\right)\tilde{\mathbf{G}}_{(n)}^{T}\left(\mathbf{C}^{(n)}\right)^{-1}$$
$$= \left(\sum_{i=S}^{E}\mathbf{X}_{(n)}^{<i>}\left(\tilde{\mathbf{A}}^{(N)}[i] \otimes \left(\otimes_{k\neq n}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)\right)\right)\tilde{\mathbf{G}}_{(n)}^{T}\left(\mathbf{C}^{(n)}\right)^{-1}$$

Next, we express $i$-th block matrix $\mathbf{X}_{(n)}^{<i>}$ as the result $(\mathbf{A}^{<i>})^{(n)}\mathbf{G}_{(n)}^{<i>}$ $\left((\mathbf{A}^{<i>})^{(N)T} \otimes \left(\otimes_{k\neq n}^{N-1}(\mathbf{A}^{<i>})^{(k)T}\right)\right)$ obtained in the preprocessing step.

$$\tilde{\mathbf{A}}^{(n)}$$
$$= \sum_{i=S}^{E}(\mathbf{A}^{<i>})^{(n)}\mathbf{G}_{(n)}^{<i>}\left((\mathbf{A}^{<i>})^{(N)T}\tilde{\mathbf{A}}^{(N)}[i] \otimes \left(\otimes_{k\neq n}^{N-1}(\mathbf{A}^{<i>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\right)$$
$$\times \tilde{\mathbf{G}}_{(n)}^{T}\left(\mathbf{C}^{(n)}\right)^{-1} = \left(\sum_{i=S}^{E}(\mathbf{A}^{<i>})^{(n)}(\mathbf{B}^{<i>})^{(n)}\right)\left(\mathbf{C}^{(n)}\right)^{-1}$$

Note that $\tilde{\mathbf{A}}^{(N)}[i]$ is described in Lemma 4. □

### B.4 Proof of Lemma 4

Proof. From Equation (3), we decouple $\tilde{\mathbf{X}}_{(N)}$ for updating $N$-th factor matrix. We first re-express $\tilde{\mathbf{X}}_{(N)}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)$ using temporal block tensors $\mathcal{X}^{<i>}$ for $i = S, .., E$ as follows:

$$\tilde{\mathbf{X}}_{(N)}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right) = \begin{bmatrix}\mathbf{X}_{(N)}^{<S>}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)\\ \vdots \\ \mathbf{X}_{(N)}^{<E>}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right)\end{bmatrix}$$

Then, we replace $\mathbf{X}_{(N)}^{<i>}$ with the tucker results obtained at the preprocessing phase.

$$\tilde{\mathbf{X}}_{(N)}\left(\otimes_{k=1}^{N-1}\tilde{\mathbf{A}}^{(k)}\right) \approx \begin{bmatrix}(\mathbf{A}^{<S>})^{(N)}\mathbf{G}_{(N)}^{<S>}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<S>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\\ \vdots \\ (\mathbf{A}^{<E>})^{(N)}\mathbf{G}_{(N)}^{<E>}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<E>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\end{bmatrix} \tag{10}$$

Next, we obtain the following equation by inserting the right term of the above equation into Equation (3):

$$\tilde{\mathbf{A}}^{(N)} = \begin{bmatrix}(\mathbf{A}^{<S>})^{(N)}\mathbf{G}_{(N)}^{<S>}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<S>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(N)}^{T}\\ \vdots \\ (\mathbf{A}^{<E>})^{(N)}\mathbf{G}_{(N)}^{<E>}\left(\otimes_{k=1}^{N-1}(\mathbf{A}^{<E>})^{(k)T}\tilde{\mathbf{A}}^{(k)}\right)\tilde{\mathbf{G}}_{(N)}^{T}\end{bmatrix}\left(\mathbf{C}^{(N)}\right)^{-1}$$
$$= \begin{bmatrix}(\mathbf{A}^{<S>})^{(N)}(\mathbf{B}^{<S>})^{(N)}\\ \vdots \\ (\mathbf{A}^{<E>})^{(N)}(\mathbf{B}^{<E>})^{(N)}\end{bmatrix}\left(\mathbf{C}^{(N)}\right)^{-1}$$

$(\mathbf{A}^{<S>})^{(N)}$ and $(\mathbf{A}^{<E>})^{(N)}$ are adjusted to fit to a range $[t_s, t_e]$. □

### B.5 Proof of Theorem 1

Proof. We split a tensor $\mathcal{X}$ into $B$ temporal block tensors $\mathcal{X}^{<i>}$, and then perform Tucker decomposition of $\mathcal{X}^{<i>}$ for $i = 1, ..., B$. Since we use Tucker-ALS in the preprocessing phase, the time complexity for each temporal block tensor $\mathcal{X}^{<i>}$ is $\mathcal{O}(MNI^{N-1}Jb)$. Therefore, the preprocessing phase takes $\mathcal{O}(MNI^{N-1}JbB)$ time. □

### B.6 Proof of Theorem 2

Proof. The time complexity of the query phase depends on updating factor matrices and core tensor. Updating a factor matrix or core tensor takes $\mathcal{O}\left(J^2 l_{[t_s,t_e]}\left(1 + \frac{NI}{b} + \frac{NJ^{N-1}}{b}\right)\right)$ time. Therefore, the total time complexity is $\mathcal{O}\left(MNJ^2 l_{[t_s,t_e]}\left(1 + \frac{NI}{b} + \frac{NJ^{N-1}}{b}\right)\right)$ which contains the time complexity of updating factor matrices and core tensor, the number of iterations, and the number of factor matrices. □

### B.7 Proof of Theorem 3

Proof. For the mode-$N$, summing up the size of the factor matrices of the time dimension is equal to $I_N J$. For each mode $n \neq N$, there are $B$ factor matrices, for the $n$-th mode, of size $\mathcal{O}(IJ)$ where $B = \frac{I_N}{b}$ is the number of blocks. Then, the space complexity is $\mathcal{O}(NIJ(\lceil \frac{I_N}{b} \rceil) + I_N J)$. □

### B.8 Proof of Theorem 4

Proof. Given a time range $[t_s, t_e]$, summing up the size of the factor matrices of the time dimension is equal to $l_{[t_s,t_e]} \times J$; the size of the factor matrix of a non-temporal mode is $I \times J$, and the number of block is equal to $\lceil \frac{l_{[t_s,t_e]}}{b} \rceil$ or $(\lceil \frac{l_{[t_s,t_e]}}{b} \rceil) + 1$. The size of the block results used in the query phase is $\mathcal{O}(NIJ(\lceil \frac{l_{[t_s,t_e]}}{b} \rceil) + l_{[t_s,t_e]}J)$. By carefully stitching the block results, intermediate data are always smaller than the block results. Therefore, the space cost of Zoom-Tucker is $\mathcal{O}\left(NIJ(\lceil \frac{l_{[t_s,t_e]}}{b} \rceil) + Jl_{[t_s,t_e]}\right)$ for a given time range $[t_s, t_e]$. □

## C PARAMETERS SETTINGS

We use the following parameters.

(1) **Number of threads:** we use a single thread.
(2) **Max number of iterations:** the maximum number of iterations is set to 100.
(3) **Rank:** we set the dimensionality $J_n$ of each mode of core tensor to 10.
(4) **Choosing a time range query:** we randomly choose a start time $t_s$ of a time range, and compute $t_e = t_s + l_{[t_s,t_e]} - 1$ where $l_{[t_s,t_e]}$ is the length of the time range; we choose $l_{[t_s,t_e]}$ among the sets described in Table 3.
(5) **Block size $b$:** we set $b$ to 50 except in Section 4.4.
(6) **Tolerance:** the iteration stops when the variation of the error $\frac{\sqrt{\|\mathcal{X}\|_F^2 - \|\mathcal{G}\|_F^2}}{\|\mathcal{X}\|_F}$ [17] is less than $\epsilon = 10^{-4}$.

Other parameters for competitors are set to the values proposed in each paper. To compare the running time, we run each method 5 times, and report the average.