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

DAO-CP: Data Adaptive Online CP Decomposition

Sangjun Son · **Yongchan Park** ·
Minyong Cho · **U Kang**

Received: date / Accepted: date

Abstract How can we decompose a streaming tensor accurately and efficiently? Many real-world tensors are dynamic, that is, their size and value evolve. CP decomposition, one of the most popular tools for analyzing tensors, is not readily applicable to temporally growing tensors. As a tensor stream creates data slices in a concise period, existing methods for dynamic tensor decomposition such have focused on reducing the running time of algorithm; however, this often entails a substantial loss of accuracy. Real-world data have temporal characteristics and change throughout time. In this work, we propose DAO–CP, a novel approach for accurate and efficient decomposition adaptive to data changes in higher-order streaming tensors. DAO–CP detects each data slice's theme and performs accuracy optimization by choosing whether to (1) split the tensor or (2) refine its factors with re-decomposition. The split process enables memory-efficient management against excessive computations due to the time-incremental factor, while the refinement process achieves much more accurate decomposition. Change detection by tracking every local error norm of each data slice determines which process to be executed. Overall decision-making module in an updatable tensor stream framework improves DAO–CP to have better efficiency and scalability. Though our experiments, DAO–CP shows higher fitness up to 91.2% while Full–CP shows 83.9% accuracy, unholding time and memory efficiency as an online method.

Keywords Tensor stream · Online CP decomposition · Drastic data change

Sangjun Son
Seoul National University,
E-mail: lucetre@snu.ac.kr

Yongchan Park
Seoul National University,
E-mail: wjdakf3948@gmail.com

Minyong Cho
Seoul National University,
E-mail: minyongcho@snu.ac.kr

U Kang
Seoul National University,
E-mail: ukang@snu.ac.kr

1 Introduction

How can we efficiently find latent patterns of higher-order tensor streams? Multi-dimensional arrays or tensors have been widely used to model real-world data (Acar et al., 2011). Tensor decomposition plays a significant role in latent feature discovery and estimation of unobservable entries (Kang et al., 2012; Cai et al., 2015; Sedghi and Sabharwal, 2018). Tensors are classified as static or dynamic in terms of temporal dependency; a tensor is called *dynamic* if its size and value change over time (e.g., indoor sensor data), or *static* if not.

As the data stream produces numerous amounts of data every moment, it has become more important to store the decomposed result for reuse, keeping the computations tractable. Since every dynamic tensor has one or more temporal modes, newly incoming data slices are stacked along with the modes. In general, many real-world data do not have a consistent temporal pattern (Fanaee-T and Gama, 2015). For example, stock data show sharp rises and plunges according to hidden time series characteristics.

Most of the existing tensor analysis methods such as CP–ALS and HOSVD have focused on decomposing static tensors (Kolda and Bader, 2009). Every static factorization method performs many iterations until convergence, and provides more approximate decomposition close to the original tensor. In that sense, static tensor factorization methods often turn out to be inappropriate in terms of time and space efficiency. Dynamic tensor factorization methods, aiming for fast factor updates, results in a substantial loss of accuracy (Kasai, 2016; Mardani et al., 2015). Indeed, current dynamic methods (1) update only non-temporal modes with pre-calculated auxiliary matrices (Zhou et al., 2016), or (2) update whole factors with prior decomposition results ignoring the previous tensor (Song et al., 2017). Consequently, none of them provides a highly accurate decomposition for real-world datasets of inconsistent temporal patterns.

In this paper, we propose DAO–CP, a novel approach for accurate and efficient decomposition adaptive to data changes in higher-order streaming tensors. The main ideas of DAO–CP are (1) to discover latent theme of tensors in newly incoming data to detect change points of themes in tensor streams, and (2) to discover latent information in newly incoming data slice by re-decomposition processes. Through comprehensive experiments, we show that DAO–CP is a highly accurate, fast, and memory-efficient online CP decomposition method.

The contributions of our project are as follows:

– **Algorithm.** We propose DAO–CP, a fast and memory-efficient method for decomposing online streaming tensor.
– **Performance.** Extensive experiments show that DAO–CP is time-scalable, and improves decomposition accuracy up to 91.2% while Full–CP shows 83.9% accuracy.

The code and datasets are available at `https://datalab.snu.ac.kr/dao-cp`. The rest of the paper is organized as follows. In Section 2, we explain preliminaries on static and dynamic tensor decomposition. Section 3 describes our proposed method DAO–CP. We demonstrate our experimental results in Section 4. After reviewing related work in Section 5, we conclude in Section 6.

Table 1: Main symbols and their descriptions.

Notation	Definition
$\mathbf{A}$	matrix
$\mathbf{A}^\top$	transpose of $\mathbf{A}$
$\mathbf{A}^\dagger$	pseudoinverse of $\mathbf{A}$
$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$	N-th order tensor
R	rank of tensor
I_n	length of the n-th mode of a tensor
$\mathbf{A}_n \in \mathbb{R}^{I_n \times R}$	n-th mode factor matrix of a tensor
$\lVert\boldsymbol{\mathcal{X}}\rVert$	Frobenius norm of $\boldsymbol{\mathcal{X}}$
$\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i \neq n} I_i}$	mode-n unfolding matrix of $\boldsymbol{\mathcal{X}}$
$[\![\cdot]\!]$	Kruskal operator, e.g. $\boldsymbol{\mathcal{X}} \approx [\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]$
$\odot, \quad \bigodot \mathbf{A}_i$	Khatri-Rao product
$\circledast, \quad \circledast \mathbf{A}_i$	Hadamard product
$\oslash$	element-wise division

2 Preliminaries

In this section, we describe notations (summarized in Table 1) and the preliminaries of tensor and online decomposition algorithm.

2.1 Tensors

Tensors are multidimensional arrays that generalize vectors (1-order tensors) and matrices (2-order tensors) to higher orders. We denote vectors with bold lowercase letters ($\mathbf{a}$), matrices with bold capital letters ($\mathbf{A}$), and tensors with bold calligraphic letters ($\boldsymbol{\mathcal{X}}$). A tensor $\boldsymbol{\mathcal{X}}$ has N modes whose lengths are $I_1, \cdots, I_N$ respectively. A tensor can be unfolded or matricized along any of its modes, and the unfolded matrix of $\boldsymbol{\mathcal{X}}$ along the n-th mode is denoted as $\mathbf{X}_{(n)}$. We denote Frobenius norm of a vector, matrix, and tensor as $\lVert\cdot\rVert$ which is defined as follows:

$$\lVert\boldsymbol{\mathcal{X}}\rVert = \sqrt{\sum_{\forall (i_1, \cdots, i_N) \in \boldsymbol{\mathcal{X}}} \left(\boldsymbol{\mathcal{X}}_{(i_1, \cdots, i_N)}\right)^2} \tag{1}$$

Hadamard product $\mathbf{A} \circledast \mathbf{B}$ and Khatri-Rao product $\mathbf{A} \odot \mathbf{B}$ are two essential matrix products used in tensor decomposition. The Hadamard product is simply an element-wise product of two matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size. The Khatri-Rao product is column-wise Kronecker product:

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1, \cdots, \mathbf{a}_J \otimes \mathbf{b}_J] \in \mathbb{R}^{I_A I_B \times J} \tag{2}$$

where $\{\mathbf{a}_n\}$ and $\{\mathbf{b}_n\}$ are the column vectors of $\mathbf{A} \in \mathbb{R}^{I_A \times J}$ and $\mathbf{B} \in \mathbb{R}^{I_B \times J}$.

Unfolding reorders elements of a tensor into a matrix (Oh et al., 2019). The mode-n unfolding matrix $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i \neq n} I_i}$ of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ maps an element $(i_1, \cdots, i_N)$ of $\mathcal{X}$ to an element (i_n, j) of $\mathbf{X}_{(n)}$ as follows:

$$j = 1 + \sum_{k=1, k\neq n}^{N} \left[(i_k - 1) \prod_{m=1, m\neq n}^{k-1} I_m \right] \tag{3}$$

2.2 Tensor Decomposition

CANDECOMP/PARAFAC (CP) decomposition is one of the most widely used methods for tensor decomposition, which is considered as a key building block in many other variants (Kolda and Bader, 2009; Sidiropoulos et al., 2017). CP decomposition factorizes a tensor into a sum of component rank-one tensors. The number of rank-one vector sets is called the rank R.

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_1^{(r)} \circ \cdots \circ \mathbf{a}_N^{(r)} \tag{4}$$

The *factor matrices* $\{\mathbf{A}_1, \cdots, \mathbf{A}_N\}$ refer to the combination of the vectors from the rank-one components.

$$\left\{ \left[\mathbf{a}_1^{(1)}, \cdots, \mathbf{a}_1^{(R)} \right], \cdots, \left[\mathbf{a}_N^{(1)}, \cdots, \mathbf{a}_N^{(R)} \right] \right\} \tag{5}$$

The vectors used in outer products form N factor matrices $\{\mathbf{A}_n \in \mathbb{R}^{I_n \times R}\}$. We express the CP decomposition result of $\mathcal{X}$ using Kruskal operator $[\![\cdot]\!]$ and the unfolding matrix, where the Kruskal operator provides a shorthand notation for the sum of outer products of the columns in factor matrices (Kolda, 2006):

$$\mathcal{X} \approx [\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!] \tag{6}$$

$$\mathbf{X}_{(n)} \approx \mathbf{A}_n \left(\bigodot_{k\neq 1} \mathbf{A}_k \right)^{\top} \tag{7}$$

Then, CP decomposition aims to find the factor matrices that minimize the estimation error $\mathcal{L}$ defined as follows:

$$\mathcal{L}(\mathbf{A}_1, \cdots, \mathbf{A}_N) = \|\mathcal{X} - [\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]\|^2 = \left\| \mathbf{X}_{(n)} - \mathbf{A}_n \left(\bigodot_{k\neq 1} \mathbf{A}_k \right)^{\top} \right\|^2 \tag{8}$$

$$\underset{\{\mathbf{A}_n\}}{\text{minimize}} \ \mathcal{L}(\mathbf{A}_1, \cdots, \mathbf{A}_N) \tag{9}$$

CP–ALS algorithm has been extensively used for this optimization problem. The main idea of alternating least squares (ALS) is to divide the original problem into N sub-problems. Specifically, each sub-problem corresponds to updating one factor matrix while keeping all the others fixed:

$$\begin{aligned} \mathbf{A}_n &\leftarrow \arg\min_{\mathbf{A}_n} \left\| \mathbf{X}_{(n)} - \mathbf{A}_n \left(\bigodot_{k\neq 1} \mathbf{A}_k \right)^{\top} \right\|^2 \\ &= \mathbf{X}_{(n)} \left(\bigodot_{k\neq 1} \mathbf{A}_k \right)^{\dagger} = \frac{\mathbf{X}_{(n)} \left(\bigodot_{k\neq n} \mathbf{A}_k \right)}{\circledast_{k\neq n} \mathbf{A}_k^{\top} \mathbf{A}_k} \end{aligned} \tag{10}$$

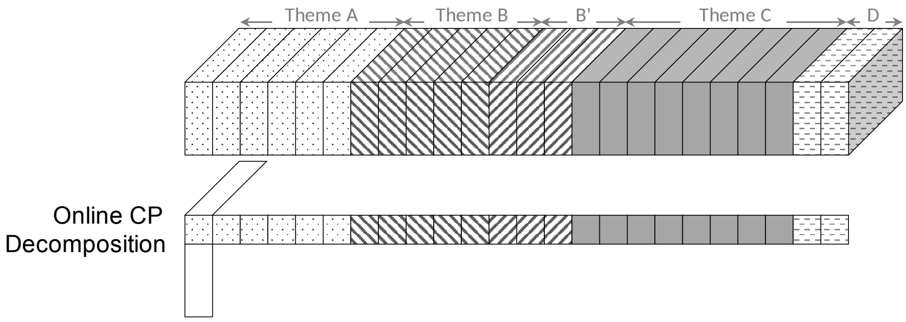


Fig. 1: Visualization of Online Tensor Decomposition: The size of the time factor becomes larger for each data income. On the contrary to static decomposition, dynamic methods take advantage of being able to reduce computation through assumptions or approximation. What if OnlineCP meets a slice of unseen theme, e.g., from theme A to B? The method updates only past factors and the current piece without saving the existing tensors due to the online setting. High accurate decomposition cannot be achieved without considering the data characteristics.

2.3 Online Tensor Decomposition

How do we decompose a streaming tensor accumulated by tensor stream? We think of a tensor consisted of a lot of slices given at each time step as shown in Fig. 1. Given an N-order temporally growing tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, we expand as a form of $\mathcal{X}^{old} \in \mathbb{R}^{I_1^{old} \times \cdots \times I_N}$ appending a new slice of data $\mathcal{X}^{new} \in \mathbb{R}^{I_1^{new} \times \cdots \times I_N}$ on its first mode. Dealing with online tensor system, we assume that $I_1^{new} \ll I_1^{old}$. How can we decompose a new tensor $\mathcal{X}^{new}$ with decomposition result $\mathcal{X}^{old} \approx [\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$ from the previous time step? Minimizing estimation error $\mathcal{L}$ is to decompose the original tensor $\mathcal{X}$ as below.

$$\mathcal{X} = \begin{bmatrix} \mathcal{X}^{old} \\ \mathcal{X}^{new} \end{bmatrix} \approx \left[\!\!\left[\begin{bmatrix} \mathbf{A}_1^{(0)} \\ \mathbf{A}_1^{(1)} \end{bmatrix}, \cdots, \mathbf{A}_N \right]\!\!\right] \quad s.t. \quad \mathbf{A}_1^{(0)} \in \mathbb{R}^{I_1^{old} \times R},\ \mathbf{A}_1^{(1)} \in \mathbb{R}^{I_1^{new} \times R}. \tag{11}$$

$$\mathcal{L} = \frac{1}{2} \left\| \mathcal{X}^{old} - [\![\mathbf{A}_1^{(0)}, \cdots, \mathbf{A}_N]\!] \right\|^2 + \frac{1}{2} \left\| \mathcal{X}^{new} - [\![\mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_N]\!] \right\|^2 \tag{12}$$

3 Proposed Method

In this section, we describe DAO–CP, our proposed online CP–ALS factorization algorithm adaptive to changing data.

3.1 Overview

DAO–CP provides a memory-efficient algorithm for fast and accurate online CP–ALS tensor decomposition considering data change. The main challenge of decomposing dynamic tensors is improving computational cost and memory usage without sacrificing accuracy. Many existing methods approximate current decomposition with previous factors for the fast computation; this, however, leads to a significant loss in accuracy. Considering that the themes of datasets change over time, we propose detecting the change points of the themes and using different strategies depending on the degree of change. There are several challenges in designing an optimized algorithm for online decomposition adaptive to data.

1. **Minimize the computational cost** How to minimize space and time cost to update overall decomposition factors when a new tensor slice is stacked?
2. **Maximize the decomposition accuracy** How to detect different themes of slices, find their latent characteristics, and eventually utilize them to improve factorization accuracy?

To overcome the above challenges, we suggest the following main ideas, which we describe in later subsections.

1. **Build an updatable framework for tensor stream** We reduce unnecessary operations by utilizing complementary matrices and previous decomposition results. Introduction of memory rate enhances refining decomposition of the current tensor with higher accuracy.
2. **Detect data changes and re-decompose data slices** We continuously track error norm for incoming data slices in the decomposition stage, detecting a sudden accuracy drop which we regard as a change point of themes. Once a sudden change is detected, we choose to refine or split the streaming tensors to factorize again and ultimately minimize global error norm.

3.2 Updatable Framework for Tensor Stream

Decomposition dealing with drastic data changes should be receptive to newly incoming slices and persistent with previous decomposition results. We choose the estimation error $\mathcal{L}$ as equation (27), which is used in DTD in MAST, and modify settings for fully observable 1st-mode streaming tensor. We simplify the online tensor decomposition problem to purely focus on maximizing performance while handling temporally changing characteristics of real-world datasets. Similar to the forgetting factor, memory rate $\mu \in [0, 1]$ is introduced for inheritance from the last time step, and also refers to acceptance for changes.

We handle this optimization problem in an alternating update fashion. For clarity of representation, we define a third order tensor $\mathcal{X} \approx \left[\!\left[\left[\mathbf{A}^{(0)}; \mathbf{A}^{(1)} \right], \mathbf{B}, \mathbf{C} \right]\!\right]$ consisting of $\mathcal{X}^{old} \approx \left[\!\left[\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}} \right]\!\right]$ to which $\mathcal{X}^{new}$ is appended.

$$\mathcal{L} = \frac{\mu}{2}\left\|\llbracket\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}}\rrbracket - \llbracket\mathbf{A}^{(0)},\mathbf{B},\mathbf{C}\rrbracket\right\|^2 + \frac{1}{2}\left\|\mathcal{X}^{new} - \llbracket\mathbf{A}^{(1)},\mathbf{B},\mathbf{C}\rrbracket\right\|^2 \tag{13}$$

$$\mathbf{A}^{(0)} \leftarrow \frac{\widetilde{\mathbf{A}}(\widetilde{\mathbf{C}}^\top\mathbf{C} \circledast \widetilde{\mathbf{B}}^\top\mathbf{B})}{\mathbf{C}^\top\mathbf{C} \circledast \mathbf{B}^\top\mathbf{B}} \tag{14}$$

$$\mathbf{A}^{(1)} \leftarrow \frac{\mathcal{X}^{new}_{(1)}(\mathbf{C} \odot \mathbf{B})}{\mathbf{C}^\top\mathbf{C} \circledast \mathbf{B}^\top\mathbf{B}} \tag{15}$$

$$\mathbf{B} \leftarrow \frac{\mu\widetilde{\mathbf{B}}(\widetilde{\mathbf{C}}^\top\mathbf{C} \circledast \widetilde{\mathbf{A}}^\top\mathbf{A}) + \mathcal{X}^{new}_{(2)}\left(\mathbf{C} \odot \mathbf{A}^{(1)}\right)}{(\mathbf{C}^\top\mathbf{C}) \circledast (\mu{\mathbf{A}^{(0)}}^\top\mathbf{A}^{(0)} + {\mathbf{A}^{(1)}}^\top\mathbf{A}^{(1)})} \tag{16}$$

$$\mathbf{C} \leftarrow \frac{\mu\widetilde{\mathbf{C}}(\widetilde{\mathbf{B}}^\top\mathbf{B} \circledast \widetilde{\mathbf{A}}^\top\mathbf{A}) + \mathcal{X}^{new}_{(3)}\left(\mathbf{B} \odot \mathbf{A}^{(1)}\right)}{(\mathbf{B}^\top\mathbf{B}) \circledast (\mu{\mathbf{A}^{(0)}}^\top\mathbf{A}^{(0)} + {\mathbf{A}^{(1)}}^\top\mathbf{A}^{(1)})} \tag{17}$$

Update rules for higher dimensional streaming tensor are derived as,

$$\mathbf{A}_1^{(0)} \leftarrow \frac{\widetilde{\mathbf{A}}_1\left(\circledast_{k\neq 1}\widetilde{\mathbf{A}}_k^\top\mathbf{A}_k\right)}{\circledast_{k\neq 1}\mathbf{A}_k^\top\mathbf{A}_k} \tag{18}$$

$$\mathbf{A}_1^{(1)} \leftarrow \frac{\mathcal{X}^{new}_{(1)}\left(\bigodot_{k\neq 1}\mathbf{A}_k\right)}{\circledast_{k\neq 1}\mathbf{A}_k^\top\mathbf{A}_k} \tag{19}$$

$$\mathbf{A}_{i\neq 1} \leftarrow \frac{\mu\widetilde{\mathbf{A}}_i\left(\circledast_{k\neq 1,i}\widetilde{\mathbf{A}}_k^\top\mathbf{A}_k\right) \circledast \widetilde{\mathbf{A}}_1^\top\mathbf{A}_1^{(0)} + \mathcal{X}^{new}_{(i)}\left(\bigodot_{k\neq 1,i}\mathbf{A}_k\right) \odot \mathbf{A}_1^{(1)}}{\left(\circledast_{k\neq 1,i}\mathbf{A}_k^\top\mathbf{A}_k\right) \circledast \left(\mu{\mathbf{A}_1^{(0)}}^\top\mathbf{A}_1^{(0)} + {\mathbf{A}_1^{(1)}}^\top\mathbf{A}_1^{(1)}\right)} \tag{20}$$

Directly calculating repetitive terms in every iteration is computationally expensive. We introduce complementary matrices $\mathbf{G}$ and $\mathbf{H}$ to reduce redundant computations.

$$\mathbf{G} = \circledast_{k\neq 1}\widetilde{\mathbf{A}}_k^\top\mathbf{A}_k, \quad \mathbf{H} = \circledast_{k\neq 1}\mathbf{A}_k^\top\mathbf{A}_k \tag{21}$$

The optimization process is based on ALS, where the complementary matrices $\mathbf{G}$ and $\mathbf{H}$ are updated whenever non-temporal factors are changed (algorithm 1).

Algorithm 1: DAO–CP Alternating Least Square (DAO-CP-ALS)

Input: factors from old tensors $\llbracket\widetilde{\mathbf{A}}_1,\cdots,\widetilde{\mathbf{A}}_N\rrbracket$, a new tensor slice $\mathcal{X}^{new}$, memory rate μ, number of ALS iterations n_{iter}

Output: updated factors $\llbracket\mathbf{A}_1,\cdots,\mathbf{A}_N\rrbracket$

Initialize complementary matrices $\mathbf{G}$ and $\mathbf{H}$ by (21)

for $n \leftarrow 1$ *to* n_{iter} **do**

  Update $\mathbf{A}_1^{(1)}$ using equation (19)   `// latter part of temporal factor` $\mathbf{A}_1$

  for $\mathbf{A}_{i\neq 1} \in$ *non-temporal factors* **do**

    Update $\mathbf{A}_i$, $\mathbf{G}$ and $\mathbf{H}$ using equation (20)

  Update $\mathbf{A}_1^{(0)}$ using equation (18)   `// former part of temporal factor` $\mathbf{A}_1$

Table 2: Execution criteria for split and refinement processes. Tracking local error norm by z-score limit automatically detects a change of themes in a tensor stream. By having two different limits, we specify which re-decomposition process is better to maximize accuracy while maintaining time and memory efficiency. Z-score limit L_s splits and initializes the decomposition, while L_r decomposes with more updates.

Process	**Trigger Condition**
Split	$z > L_s$
Refinement	$L_r < z \leqslant L_s$
-	$z \leqslant L_r$

3.3 Change Detection with Local Error Norm

Since dynamic tensor decomposition pursues shorter time factor updates, limited ALS iterations result in low accuracy factorization when real-time data are stacked. To maximize the decomposition accuracy, which means to minimize estimation error $\mathcal{L}$ in equation (27), we have to additionally refine mis-decomposed tensor slices.

When decomposing a tensor, error norm is a fundamental metric to compare real and estimated entries. As the tensor temporally grows, we define local and global error norm by measuring regions as a data slice and the whole tensor respectively. Mis-decomposed slices are detected by measuring local error norm for current incoming tensor and distribution of previous norms.

$$\mathcal{E}_{local} = \left\| \mathcal{X}^{new} - \left[\!\left[\mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_N \right]\!\right] \right\|^2 \sim \mathcal{N}(\mu, \sigma^2) \tag{22}$$

We assume error norm $\mathcal{E}_{local}$ follows a normal distribution and keep track of their mean and variance to find out when outliers are detected. Since the proposed method, DAO–CP is an online algorithm, updates of mean and variance should also be online. Welford's algorithm from (Welford, 1962), one of the most popular methods that compute sample means and variances, is a simple iterative method that requires only one pass of the data (Ling, 1974). It provides accurate estimates of mean and variance without having to keep the entire data.

Using Welford's algorithm, we easily find out anomalies in the current local error norm by z-score calculation with online-tracked mean and variance. Changing the limit L of z-score in equation (23), it is possible to determine the criterion on whether a tensor is well-decomposed or not.

$$z = \frac{\mathcal{E}_{local} - \mu}{\sigma} > L \tag{23}$$

3.4 Handling Drastically Changing Data

Tracking the distribution of local error norm and defining the z-score limit enable DAO–CP to trigger one of accuracy optimizing processes; the trigger function

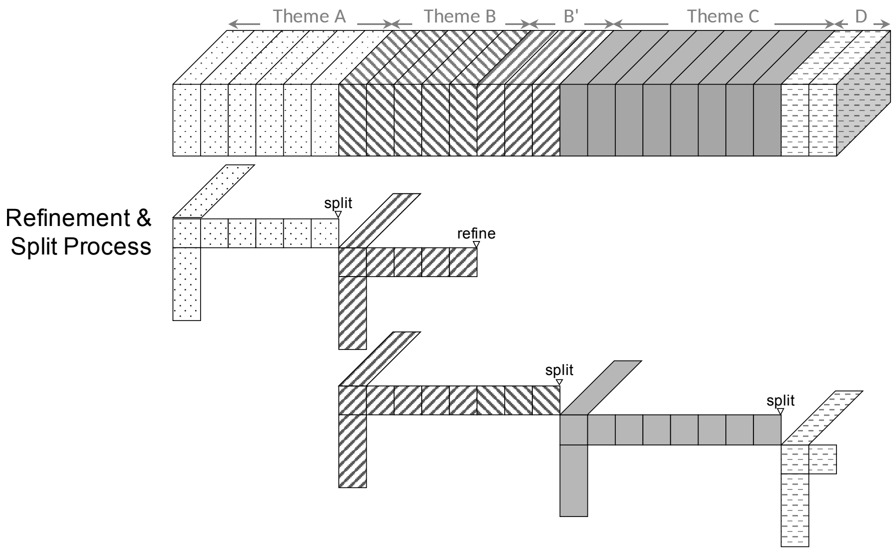


Fig. 2: Visualization of Refinement and Split process of DAO–CP: As a new slice of theme B is stacked on tensors of theme A, our method performs DTD, which is a baseline method of dynamic decomposition.It calculates the z-score in the local error norm distribution and re-decompose if detected anomalies above the threshold. When exceeding the split threshold L_s, a transition from A to B, DAO–CP splits and restarts decomposition by initialization with Full–CP. The similar theme B' is detected when the z-score exceeds L_r but not as much as the split threshold. The refinement process re-decomposes the current tensor slice more receptively with a lower memory rate.

decides to execute split or refinement process to differentiate the tensors with different themes or to gather similar tensors. L_s and L_r refer to the trigger limit for each optimization as shown in Table 2.

3.4.1 Refinement Process

Refinement process is to update incoming tensor slices whose theme is similar to the previous tensor. Memory rate μ is set to a value close to zero, and ALS operation is performed several times. It evokes a decreasing effect of previous factors on the loss function $\mathcal{L}$ in equation (27). Exceedance of z-score limit for refinement L_r triggers forgetting the previous result and decomposing the current tensor once again.

3.4.2 Split Process

Anomaly detection in local error norm tells us sudden change in data. What if incoming data slice has a different theme compared to previous tensors? It implies that a new decomposition starting point with a new split is needed. In the split process, the trigger function of threshold L_s splits the streaming tensor into serial tensors of different themes. In spite of additional consumption of space and time while initialization before online tensor updates, it brings highly improved accuracy on decomposition.

The overall process of the proposed method is shown in algorithm 2.

Algorithm 2: Data Adaptive Online CP Decomposition (DAO–CP)

Input: tensor stream $\mathcal{X}_{stream}$, memory rate μ, number of ALS iterations n_{iter}

Output: decomposition factor set $\mathcal{S} = \{[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]\}$

Get a new slice $\mathcal{X}_0 \leftarrow \mathcal{X}_{new}$ from $\mathcal{X}_{stream}$

Initialize $[\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$ using Full–CP from $\mathcal{X}_0$

Calculate error norm $\mathcal{E}_0$ between $\mathcal{X}_0$ and $[\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$

Initialize *Welford* with $\mathcal{E}_0$

for $\mathcal{X}_0$ *from* $\mathcal{X}_{stream}$ **do**

  $[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!] \leftarrow$ DAO-CP-ALS$\Big([\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!],\ \mathcal{X}_{new},\ \mu,\ n_{iter}\Big)$

  Calculate error norm $\mathcal{E}_{local}$ between $\mathcal{X}_{new}$ and $[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]$

  `/* Split Process */`

  if *Welford z-score* $(\mathcal{E}_{local}) > L_s$ **then**

    Store the previous factors to $\mathcal{S}$

    Initialize $[\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$ using Full–CP from $\mathcal{X}_{new}$

    Calculate error norm $\mathcal{E}_0$ between $\mathcal{X}_0$ and $[\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$

    Initialize *Welford* with $\mathcal{E}_0$

    continue

  `/* Refinement Process */`

  if $L_s \geqslant$ *Welford z-score* $(\mathcal{E}_{local}) > L_r$ **then**

    $[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!] \leftarrow$ DAO-CP-ALS$\Big([\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!],\ \mathcal{X}_{new},\ 1-\mu,\ 2 \cdot n_{iter}\Big)$

    Calculate error norm $\mathcal{E}_{local}$ between $\mathcal{X}_{new}$ and $[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]$

  Update *Welford* with $\mathcal{E}_{local}$

  Update $[\![\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_N]\!]$ as $[\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!]$

4 Experiments

In this section, we experimentally evaluate our proposed method DAO–CP. The experimental settings are designed to answer the following questions.

- **Q1. Reconstruction Error (Section 4.2).** How accurately do DAO–CP and other methods factorize given tensors in terms of global and local fitness?
- **Q2. Speed and Memory Efficiency (Section 4.3).** How much does DAO–CP accelerate the factorization speed and reduce the memory cost compared to OnlineCP, DTD and Full–CP?

Table 3: Datasets used to evaluate performance of DAO–CP and competitors.

Datasets	Order	Dimensions	Batch Sizes	Rank	L_s	L_r
Synthetic Data[1]	4	(1K, 10, 20, 30)	[10] * 100	30	1.2	1.1
Sample Video[2]	4	(205, 240, 320, 3)	[5] * 41	30	6.0	2.0
Stock[3]	3	(3K, 140, 5)	[3] * 1K	20	6.0	5.0
Airport Hall[4]	3	(200, 144, 176)	[10] * 20	20	0.5	0.1
Korea Air Quality[5]	3	(10K, 323, 6)	[100] * 100	20	2.0	1.3

[1] https://github.com/lucetre/online-tensor-decomposition/
[2] https://www.sample-videos.com/
[3] https://deeptrade.co/
[4] https://github.com/hiroyuki-kasai/OLSTEC/
[5] https://www.airkorea.or.kr/eng/

- **Q3. Model Parameter Adjustment (Section 4.4).** How much does the z-score limit affect the number of split and refinement processes in DAO–CP?
- **Q4. Process Validation (Section 4.5).** Does each re-decomposition process affect the performance in DAO–CP and work well on real-world datasets?
- **Q5. Time Scalability (Section 4.6).** How well does DAO–CP scale with respect to the time length of the tensor stream compared to Full–CP?

In the following, we describe the experimental settings, and answer the questions with the experimental results.

4.1 Experimental Settings

Experiments are done in a workstation with a single CPU (Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz). For the demonstration, we will use a synthetic tensor and several real-world tensors. We use 1 synthetic, and 4 real-world streaming tensors described in Table 3.

Datasets Each tensor has its first mode as a temporal mode. We construct a tensor stream by splitting into slices with batch sizes. (1) Synthetic dataset is made of concatenated tensors which is summation of three tensors T_{main}, T_{theme} and T_{noise}. Each tensor refers to 100x, 10x and 1x normal distributed randomized tensor. (2) Sample video dataset is a series of animation frames whose pixel has its RGB value. (3) Stock dataset involves 140 stocks data existing from Jan 2, 2008 to June 30, 2020 listed on Korea Stock Price Index 200 (KOSPI 200). We define features to describe the trend of stocks. Features consist of adjusted opening price divided by adjusted opening price of the day earlier, adjusted closing price divided by adjusted closing price of the day earlier, adjusted highest price divided by adjusted highest price of the day earlier, adjusted lowest price divided by adjusted lowest price of the day earlier, and trading volume over total shares listed on the stock market. (4) Airport hall dataset is made of a recorded video in an airport and was initially used to verify OLSTEC (Kasai, 2016; Zhang et al., 2017). (5) Air quality dataset is a set of daily measures in Seoul, Korea from Sep 1, 2018 to Sep 31, 2019. Measurements of pollutants are recorded varying to dates and locations.

Competitors We compare DAO–CP with dynamic methods like OnlineCP (Zhou et al., 2016), DTD (Song et al., 2017) and static method as Full–CP (Kolda and Bader, 2009). All methods including DAO–CP are implemented in Python3 using TensorLy library.

Parameters For each dataset, hyperparameters like decomposition rank affect the performance of decomposition. In DAO–CP, triggering condition for split and refinement process depends on split and refinement limit of z-score, L_s and L_r. For a fair evaluation of decomposition results, settings of model parameters that show the best performance per each dataset are also shown in Table 3.

Fitness We evaluate accuracy with local error norm $\mathcal{E}_{local}$ and global error norm $\mathcal{E}_{global}$ which we described in Section 3.3. $\mathcal{FIT}_{local}$ means fitness for an incoming data slice while $\mathcal{FIT}_{global}$ is fitness for whole tensors.

$$\mathcal{E}_{local} = \left\| \mathcal{X}^{new} - [\![\mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_N]\!] \right\|^2, \quad \mathcal{E}_{global} = \left\| \mathcal{X} - [\![\mathbf{A}_1, \cdots, \mathbf{A}_N]\!] \right\|^2 \tag{24}$$

$$\mathcal{FIT}_{local} = 1 - \frac{\mathcal{E}_{local}}{\|\mathcal{X}^{new}\|}, \quad \mathcal{FIT}_{global} = 1 - \frac{\mathcal{E}_{global}}{\|\mathcal{X}\|} \tag{25}$$

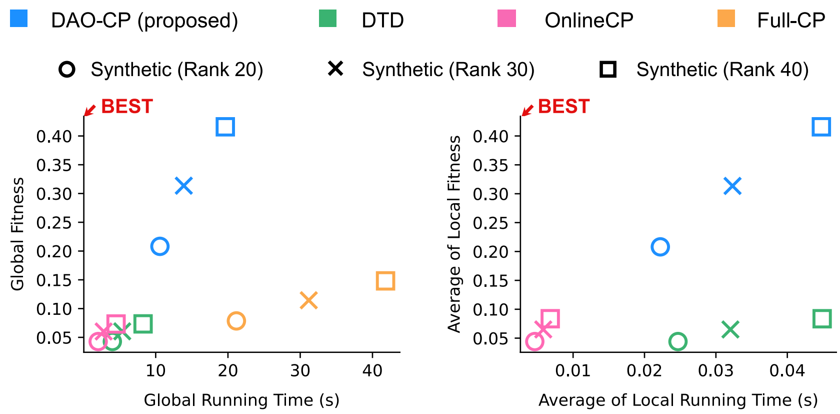
Running Time We evaluate speed metric in terms of $\mathcal{RT}_{local}$ and $\mathcal{RT}_{local}$. Local running time, $\mathcal{RT}_{local}$ indicates elapsed time for decomposing current data slice. DAO–CP includes re-decomposition time for split or refinement process. Global running time, $\mathcal{RT}_{global}$ is total elapsed time for decomposing tensor stream.

4.2 Reconstruction Error

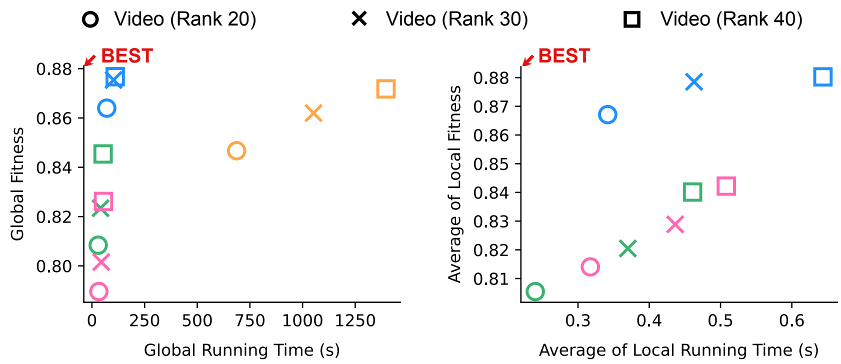
We compare DAO–CP to other dynamic or static CP decomposition methods in terms of time efficiency and fitness. In Fig. 3, DAO–CP shows higher accuracy than existing dynamic CP methods with little sacrifice in running time. Especially on Synthetic, Sample video, and Korea air quality dataset, DAO–CP is better than static/dynamic competitors in terms of both speed and accuracy. From Fig. 3b which shows visualization of decomposition of streaming video tensors, DAO–CP outputs more accurate decomposition results. Since DAO–CP well catches drastic data changes and re-decompose tensors, re-built images seem more clear than results of other methods.

4.3 Speed and Memory Efficiency

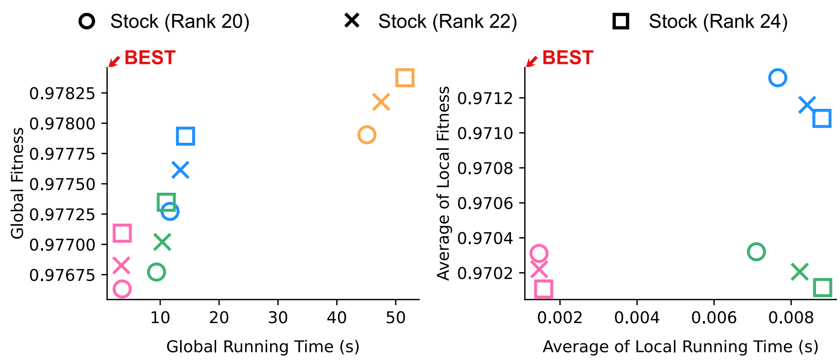
Time Cost DAO–CP gives accurate decomposition by additional processes to exploit data characteristics and detect dramatic changes. By doing so, the re-decomposition process takes a slightly longer time. In Fig. 3, DAO–CP shows weak speed on the overall decomposition process contrary to OnlineCP and DTD. According to local running time, however, DAO–CP shows similar computation speed on decomposing 1-batch-sized data slices. Compared to static methods like Full–CP, DAO–CP outperforms in time complexity and decomposes regardless of how many data slices are stacked in tensor $\mathcal{X}$ like in Fig. 6.



(a) Synthetic Data

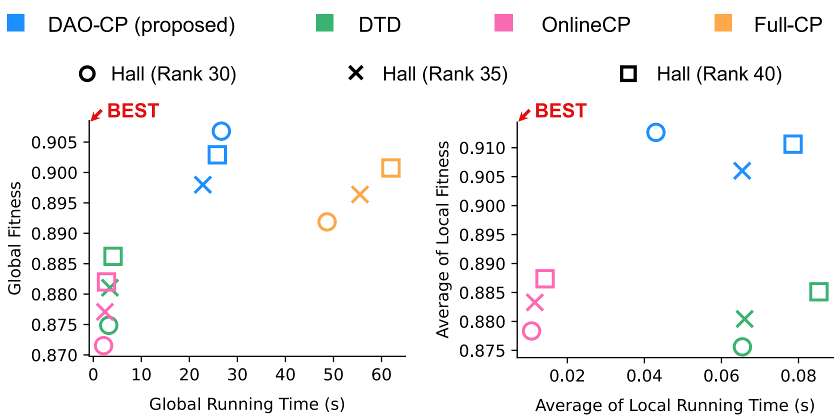


(b) Sample Video

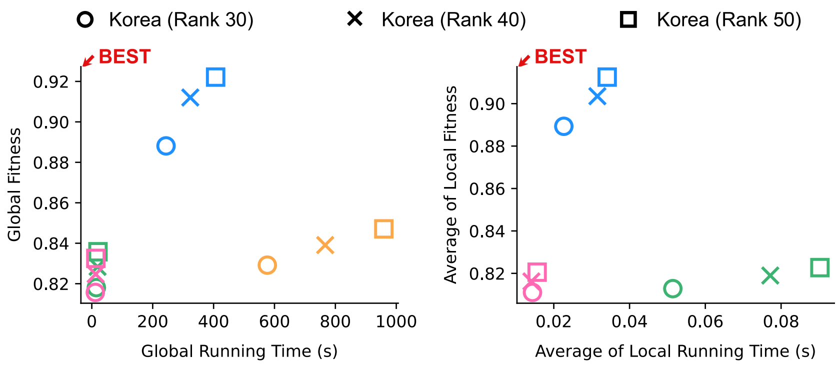


(c) Stock

Fig. 3: Time Efficiency and Fitness: global (left) and local (right) running time $\mathcal{RT}$ againt fitness $\mathcal{FIT}$.



(d) Airport Hall



(e) Korea Air Quality

Fig. 3: Time Efficiency and Fitness: global (left) and local (right) running time $\mathcal{RT}$ againt fitness $\mathcal{FIT}$. Since Full–CP is not an online method, we only evaluate global fitness and running time at the last moment when no slices are left to be stacked. DAO–CP performs re-decomposition and consumes extra computation time, but the average local speed is similar to other dynamic methods. DAO–CP enhances decomposition accuracy which is even higher than that of Full–CP for Synthetic data, Sample video, and Korea air quality datasets.

Space Cost We compare data usage while initializing, decomposing, and re-decomposing in Fig. 4. Since static methods like Full–CP do not perform in the streaming environment, intermediate memory usage is quite large because all tensors must be accumulated to decompose at once. OnlineCP, one of the fastest online cp decomposition methods, utilizes memory to initialize complementary matrices before ALS updates. In the case of DTD, on the other hand, memory risk does not occur, because the current data slice is decomposed by the previous factors, not storing the existing tensor slices. DAO–CP, which is based on DTD, allocates additional memory for factors since the tensor is continuously divided and stored by the split process.

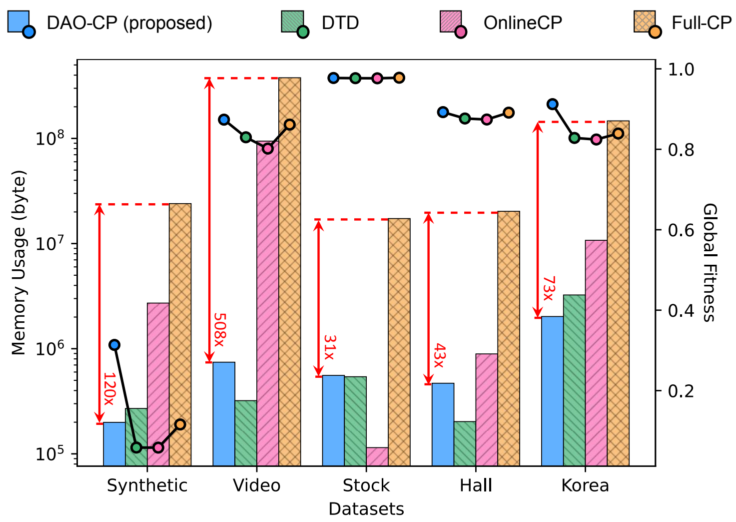


Fig. 4: Memory Efficiency: memory usage and global fitness while initializing, decomposing, and re-decomposing. We set up model parameters for each dataset in Table 3 and measure memory usage and accuracy of both static and dynamic methods. DAO–CP produces more accurate decomposition results, especially for the video dataset; space consumption of our strategy is 508x lower than that of Full–CP. Full–CP stores all the stacked tensor from stream to decompose at last, and OnlineCP has time-dependent auxiliary matrices. They result in high memory usage. DAO–CP based on DTD is memory efficient since the method only stores previous factors and the current slice for intermediate data. Yet, DAO–CP consumes a bit more memory space than DTD to keep extra decomposition factors when the split process is triggered.

4.4 Model Parameter Adjustment

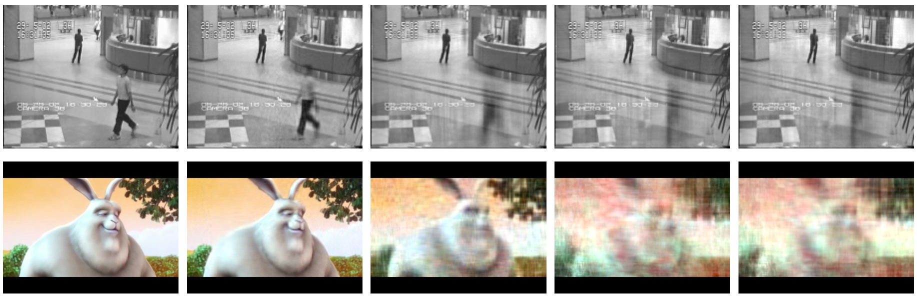
Rank As decomposition rank increases, every factorization method has larger factor matrices to tune a tensor. We reduce estimation error $\mathcal{L}$ and eventually increase decomposition accuracy. However, this leads to time and memory consumption during ALS updates. As shown in Fig. 3, both fitness $\mathcal{FIT}_{global}$, $\mathcal{FIT}_{local}$ get higher while running time $\mathcal{RT}_{global}$, $\mathcal{RT}_{local}$ elapse longer.

Z-score Limit DAO–CP has parameters L_s and L_r, which are thresholds for change detection and lead to trigger split or refinement processes. We change the values of L_s and L_r to show the effects of each process. In Table 4, number of split points changes as L_s value varies. The more parts the tensor is split into, DAO–CP gets higher accuracy with some computational cost in terms of time and memory. Similarly, the refinement process also leads to more accurate decomposition with lower speed but maintaining memory usage.

4.5 Process Validation

Table 4: Process Validation: DAO–CP performs split (upper) and refinement (lower) process whenever z-score limit detects data changes. We choose Korea air quality dataset with rank 20 and change parameter L_s and L_r to determine the number of split/refinement points. Application of re-decomposition induces remarkable accuracy improvements: 7.06% with the split process and 0.22% with the refinement process.

Process	L_s	L_r	**Number of Points**	$\mathcal{FIT}_{global}$	$\mathcal{RT}_{global}$	**Memory Usage**
None	-	-	0	**0.804990**	12.721802 sec	424 bytes
Split	1.8	-	5	0.821704	24.049187 sec	456 bytes
	1.6	-	36	0.852703	125.234840 sec	760 bytes
	1.4	-	45	0.857204	147.559270 sec	760 bytes
	1.2	-	57	0.863426	184.391101 sec	856 bytes
	1.0	-	65	**0.875591**	211.415973 sec	968 bytes
Refine	-	2.2	6	0.806233	13.253572 sec	504 bytes
	-	2.0	8	0.806506	13.393017 sec	504 bytes
	-	1.8	12	0.806698	12.561460 sec	504 bytes
	-	1.6	14	0.806959	12.481139 sec	504 bytes
	-	1.4	15	**0.807248**	12.960902 sec	504 bytes



(a) Original (b) DAO–CP (c) Full–CP (d) DTD (e) OnlineCP

Fig. 5: Process Validation: DAO–CP detects different themes of video frame and re-decompose them. To compare images at certain time frame, we decompose streaming video tensors with rank 100 from Airport hall (upper) and Sample video (lower) datasets. Original and estimated video of static and dynamic decomposition methods including DAO–CP.

The split process is to make good ends and new starts when different themes are detected and the refinement process is to re-decompose much more receptive to the current slice. As shown in Table 4, re-decomposition processes enhance accuracy with some computational issues such as time dilation and additional space costs. Fig. 5 mentions that DAO–CP creates a well-estimated tensor for the streaming video which contains several different themes and angle changes. So, both processes are indispensable in the data-adaptive algorithm, DAO–CP.

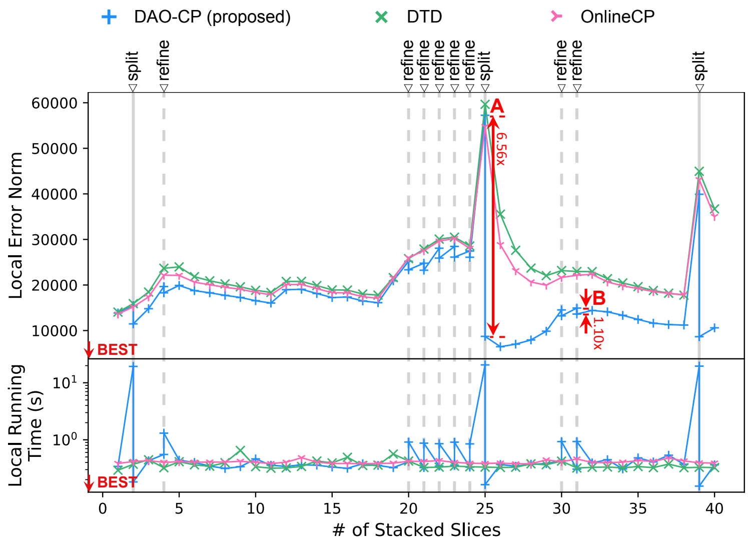


Fig. 6: Time Scalability: effects of split (solid) and refinement (dashed) process in terms of local error norm $\mathcal{E}_{local}$(upper) and running time $\mathcal{RT}_{local}$(lower). Each re-decomposition process remarkably reduces the current incoming slice's local error norm with a little sacrifice of running time. At point A, where it performs 6.56x accuracy, our method chooses the split process to initialize the decomposition. At point B, where it performs 1.10x accuracy, it selects the refinement process to re-update similar tensors. DAO–CP performs accurate decomposition regardless of temporal mode length of the stacked tensor.

4.6 Time Scalability

DAO–CP splits or refines the tensor stream to accelerate decomposition accuracy. How does the additional re-decomposition process of DAO–CP affect speed and accuracy? Is DAO–CP scalable with regard to time without any delay? Since the split process repeats stopping and restarting decomposition, it invokes heavy computation time while initializing with Full–CP. The refinement process is a simple re-decomposition that slightly costs computation time. We verify that computational costs of DAO–CP are not time-dependent due to increasing sizes of the tensor accumulated large amounts of data slices. Fig. 6 shows that DAO–CP is time-scalable although re-decomposition might induce immediate time delay.

5 Related Works

Starting from the first streaming fashioned CP decomposition showed in signal processing, many existing algorithms are in-efficient in the aspect of both memory usage and computation time and only support decomposition for 3rd order tensors (Nion and Sidiropoulos, 2009; Mardani et al., 2015). Today, streaming tensor factorization has been widely studied under the CP factorization (Smith et al., 2018; Najafi et al., 2019). And those approaches are classified into two main

Table 5: Comparison of existing online decomposition methods. (fake)

	Time Complexity	Online	Updatable
Full–CP	$\mathcal{O}\left(NR \cdot (I_1^{old} + I_1^{new}) \cdot \prod_{i \neq 1} I_i\right)$		✓
OnlineCP	$\mathcal{O}\left(NR \cdot I_1^{new} \cdot \prod_{i \neq 1} I_i\right)$	✓	
DTD	$\mathcal{O}\left(NR \cdot I_1^{new} \cdot \prod_{i \neq 1} I_i + NR^2 \cdot \sum_i I_i\right)$	✓	✓

traits: (1) update the factors for only non-temporal modes with pre-calculated auxiliary matrices, (2) update whole factors with prior decomposition results ignoring the previous tensor. We choose OnlineCP (Zhou et al., 2016) and DTD (Song et al., 2017) for each trait and compare their performance against our proposed method DAO–CP including traditional static decomposition, Full–CP. Table 5 shows comparison between the state-of-the-art decomposition methods for higher-order tensors.

5.1 OnlineCP

OnlineCP preserves the previous temporal factor to decompose the current tensor slices and efficiently decomposes the current tensor slice with time-scalability. After updates of non-temporal factors and the partial temporal factor, it simply appends the part of the temporal factor matrix to the previous matrix as shown in equation (26).

$$\mathbf{A}^{(0)} \leftarrow \widetilde{\mathbf{A}}, \quad \mathbf{A}^{(1)} \leftarrow \frac{\mathcal{X}_{(1)}^{new}\left(\mathbf{C} \odot \mathbf{B}\right)}{\mathbf{C}^\top \mathbf{C} \circledast \mathbf{B}^\top \mathbf{B}} \tag{26}$$

OnlineCP avoids duplicated computations like Khatri-Rao and Hadamard products by introducing auxiliary matrices. Using dynamic programming strategy, the method computes complementary matrices before ALS iteration, and easily tracks the new decomposition to temporally store the useful information of the previous timestep (Zhou et al., 2016).

5.2 DTD

DTD is originally introduced as a part of MAST which is a low-rank tensor completion method to fill the missing entries of the incomplete multi-aspect streaming tensor (Song et al., 2017). The method takes advantage of computational complexity reduction by one assumption; the previous decomposition approximates the tensor stacked until current data income. Since $\mathcal{X}_{old}$ approximates $[\![\widetilde{\mathbf{A}}_1, \cdots, \widetilde{\mathbf{A}}_N]\!]$, we can rewrite the estimation error of online tensor decomposition in equation (26) as below. Forgetting factor $\mu \in [0, 1]$ alleviates the influence of the previous decomposition error. The assumption enables the method to avoid heavy computations by changing Khatri-Rao to Hadamard products.

$$\mathcal{L} = \frac{\mu}{2}\left\| \left[\!\!\left[\widetilde{\mathbf{A}}_1, \cdots, \widetilde{\mathbf{A}}_N \right]\!\!\right] - \left[\!\!\left[\mathbf{A}_1^{(0)}, \cdots, \mathbf{A}_N \right]\!\!\right] \right\|^2 + \frac{1}{2}\left\| \mathcal{X}^{new} - \left[\!\!\left[\mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_N \right]\!\!\right] \right\|^2 \tag{27}$$

6 Conclusions

In this paper, we propose DAO–CP (Data Adaptive Online CP decomposition), an efficient algorithm for decomposition of time-evolving tensors. DAO–CP automatically detects the drastic changes in tensor stream and decides whether to re-decompose the tensors. The resulting algorithm is time-scalable, having computational cost only proportional to the size of incoming data. Experimental results show that DAO–CP gives better accuracy than all competitors in various real-world tensor stream datasets.

Overall decision-making module in an updatable tensor stream framework improves DAO–CP to have better efficiency and scalability. We empirically demonstrate that the proposed DAO–CP achieves comparable accuracy on both synthetic and real-world datasets in an online streaming environment. Future works include developing DAO–CP to completion task of sparse tensor stream and generalizing the problem to a multi-aspect stream condition.

References

Acar E, Dunlavy DM, Kolda TG, Mørup M (2011) Scalable tensor factorizations for incomplete data. Chemometrics and Intelligent Laboratory Systems 106(1):41–56

Battaglino C, Ballard G, Kolda TG (2018) A practical randomized cp tensor decomposition. SIAM Journal on Matrix Analysis and Applications 39(2):876–901

Cai Y, Tong H, Fan W, Ji P, He Q (2015) Facets: Fast comprehensive mining of coevolving high-order time series. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 79–88

Fanaee-T H, Gama J (2015) Multi-aspect-streaming tensor analysis. Knowledge-Based Systems 89:332–345

Kang U, Papalexakis E, Harpale A, Faloutsos C (2012) Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 316–324

Kasai H (2016) Online low-rank tensor subspace tracking from incomplete data by cp decomposition using recursive least squares. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp 2519–2523

Kolda TG (2006) Multilinear operators for higher-order decompositions. Tech. rep., Sandia National Laboratories

Kolda TG, Bader BW (2009) Tensor decompositions and applications. SIAM review 51(3):455–500

Ling RF (1974) Comparison of several algorithms for computing sample means and variances. Journal of the American Statistical Association 69(348):859–866

Mardani M, Mateos G, Giannakis GB (2015) Subspace learning and imputation for streaming big data matrices and tensors. IEEE Transactions on Signal Processing 63(10):2663–2677

Najafi M, He L, Philip SY (2019) Outlier-robust multi-aspect streaming tensor completion and factorization. In: IJCAI, pp 3187–3194

Nion D, Sidiropoulos ND (2009) Adaptive algorithms to track the parafac decomposition of a third-order tensor. IEEE Transactions on Signal Processing 57(6):2299–2310

Oh S, Park N, Jang JG, Sael L, Kang U (2019) High-performance tucker factorization on heterogeneous platforms. IEEE Transactions on Parallel and Distributed Systems 30(10):2237–2248

Sedghi H, Sabharwal A (2018) Knowledge completion for generics using guided tensor factorization. Transactions of the Association for Computational Linguistics 6:197–210

Sidiropoulos ND, De Lathauwer L, Fu X, Huang K, Papalexakis EE, Faloutsos C (2017) Tensor decomposition for signal processing and machine learning. IEEE Transactions on Signal Processing 65(13):3551–3582

Smith S, Huang K, Sidiropoulos ND, Karypis G (2018) Streaming tensor factorization for infinite data sources. In: Proceedings of the 2018 SIAM International Conference on Data Mining, SIAM, pp 81–89

Song Q, Huang X, Ge H, Caverlee J, Hu X (2017) Multi-aspect streaming tensor completion. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 435–443

Welford B (1962) Note on a method for calculating corrected sums of squares and products. Technometrics 4(3):419–420

Zhang S, Guo B, Dong A, He J, Xu Z, Chen SX (2017) Cautionary tales on air-quality improvement in beijing. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 473(2205):20170457

Zhou S, Vinh NX, Bailey J, Jia Y, Davidson I (2016) Accelerating online cp decompositions for higher order tensors. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 1375–1384