Online CP Decomposition for Sparse Tensors

Shuo Zhou, Sarah M. Erfani and James Bailey
School of Computing and Information Systems, The University of Melbourne, Australia
{zhous@student., sarah.erfani@, baileyj@}unimelb.edu.au

***Abstract*—Tensor decomposition techniques such as CANDECOMP/PARAFAC (CP) decomposition have achieved great success across a range of scientific fields. They have been traditionally applied to dense, static data. However, today's datasets are often highly sparse and dynamically changing over time. Traditional decomposition methods such as Alternating Least Squares (ALS) cannot be easily applied to sparse tensors, due to poor efficiency. Furthermore, existing online tensor decomposition methods mostly target dense tensors, and thus also encounter significant scalability issues for sparse data. To address this gap, we propose a new incremental algorithm for tracking the CP decompositions of online sparse tensors on-the-fly. Experiments on nine real-world datasets show that our algorithm is able to produce quality decompositions of comparable quality to the most accurate algorithm, ALS, whilst at the same time achieving speed improvements of up to 250 times and 100 times less memory.**

I. Introduction

Multi-dimensional datasets are common in a wide range of applications such as chemometrics [1], signal processing [2], machine learning [3] and data mining [4]. A natural choice for representing such data is a *Tensor*, which is a multi-way array that can preserve the complex relationships among different dimensions. *Tensor Decomposition* (TD) is a fundamental technique for analyzing tensors, and it has been extensively studied and widely applied for varying tasks [5]. However, existing TD techniques are usually designed for dense and static tensors, which makes them less suitable when the data is highly sparse and dynamically changing over time.

Many real-world tensors are very large and have high sparsity. Thus, compared to their overall size, the number of non-zero entries is small. As a consequence, it is difficult to apply existing TD techniques, since these have often been designed for dense tensors, and thus have poor efficiency and scalability when applied to sparse scenarios. Moreover, sparse tensors are often not static. Instead, they are often changing over time, as large volumes of new data is generated and added to the the existing tensor. One typical type of dynamic update is appending new data along a specific dimension such as time, with the other dimensions remaining unchanged. An example is time-evolving social networks, as shown in Figure 1. The network can be represented by a $user \times user \times time$ tensor and each slice at its time dimension is a collection of user interactions such as tagging a friend in a photo or visiting a friend's homepage. Overall, this tensor is highly sparse by its nature, and rapidly growing as new data (slices) are added. We refer to such tensors as *online sparse tensors*.

The research problem we address in this paper is how to efficiently decompose online sparse tensors. We are particularly

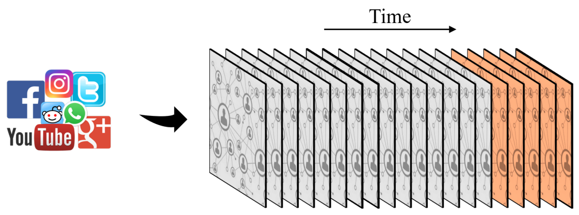

Fig. 1. An example of online sparse tensors

interested in CANDECOMP/PARAFAC (CP) decomposition, since it is one of the most popular TD techniques having numerous applications. Specifically, given the existing CP decomposition of a sparse tensor and a batch of new sparse data slices, which get added to the tensor's time mode, *we would like to efficiently obtain the new CP decomposition of the newly formed tensor without computing it from scratch.*

To the best of our knowledge, this is an unresolved question and existing approaches are not suitable for decomposing online sparse tensors. Alternating Least Squares (ALS) has been widely considered as the workhorse for CP decomposition [5] due to its simplicity and ease of implementation. However, it is inapplicable to decompose an online large scale sparse tensor, due to its poor efficiency. Even though distributed and parallel algorithms [6], [7], [8] may be used to accelerate ALS, the number of non-zeros in the new data can be too small, so it is wasteful to employ computational resources to decompose the full tensor from scratch. Lastly, there are existing approaches that have been designed for incremental, online decompositions [9], [10], but these are designed for online *dense* tensors and their complexities are usually linear to the size of new data. This significantly limits their applicability to online sparse tensors which can be very large, but contain few non-zero elements.

Overall, due to their poor efficiency and scalability, existing methods are not well suited for decomposing online sparse tensors. To address this gap, we propose a new algorithm, OnlineSCP. The contributions of our work are as follows:

- We propose a new algorithm having linear complexity to the number of non-zeros in the new data, for tracking the CP decomposition of an online sparse tensor.
- Via experiments on nine real-world datasets, our method demonstrates high decomposition quality, as well as significant efficiency improvements in both time and memory usage.

II. Related Work

In order to decompose a large-scale sparse tensor efficiently, Bader and Kolda [11] proposed an algorithm based on ALS, tailoring it to sparse tensors, with support from special data structures and customized tensor-related operations. Following the same idea, Smith *et al.* [12] proposed a new hierarchical data structure called a Compressed Sparse Fiber (CSF), which is more memory-efficient and much easier to parallelize. Li *et al.* [13] also applied CSF as the storage format for sparse tensors, while a variant of CSF (vCSF) was used for recording intermediate results, which leads to more memory-savings and speedup. In addition, with advances in distributed and paralleled computing, techniques and tools like MapReduce [6], [7], [8] and GPUs [14] have also been used to further speed up ALS for large scale sparse tensors.

In terms of online tensor decomposition, the problem was first proposed by Sun *et al.* [15], [16], wherein they refer to this problem as *Incremental Tensor Analysis* (ITA). However, their algorithms are online versions of Tucker decomposition. Although CP decomposition can be viewed as a special case of Tucker with super-diagonal core tensor, there is no way to enforce this constraint can be found. As a result, these algorithms are not suitable for tracking the CP decompositions of online sparse tensors.

Limited research can be found for online CP decompositions. An early exploration was done by Nion and Sidiropoulos [9], introducing two adaptive algorithms that specifically focus on CP decomposition: Simultaneous Diagonalization Tracking (SDT), which incrementally tracks the SVD of the unfolded tensor; and Recursive Least Squares Tracking (RLST), which recursively updates the decomposition factors by minimizing the mean squared error. However, the limitation of this work is that it can only be applied to third-order tensors. Zhou *et al.* [10] proposed a general approach that incrementally tracks the CP decompositions of online tensors with arbitrary dimensions. However, both [9] and [10] are specifically designed for dense online tensors, which means that their efficiency quickly drops to an unacceptable level and may potentially run out of memory for sparse online tensors having large non-temporal modes. A more recent technique proposed by Gujral *et al.* [17] can operate with both dense and sparse online tensors via sub-sampling, though multiple repetitions are required for a stable result.

Lastly, it is worth mentioning studies that address related, but somewhat different questions to our research problem. Zhou *et al.* [18] proposed a dynamic learning schema for tracking CP decomposition of sparse tensors that are incrementally changing at the element (not slice) level. Shin *et al.* [19] also studied dynamic sparse tensors with element-wise updating and proposed an efficient algorithm for dense-subtensor detection. Song *et al.* [20] addressed the tensor completion problem based on CP decomposition for dynamic tensors that are growing across all modes (rather than just a single mode, which will be our focus).

TABLE I
List of symbols

a, $\mathbf{a}$, $\mathbf{A}$, $\mathcal{X}$	scalar, vector, matrix, tensor
$\mathbf{a}_{i\cdot}$, $\mathbf{a}_{\cdot j}$	the i-th row and j-th column vectors of $\mathbf{A}$
a_{ij}	the (i, j)-th element of $\mathbf{A}$
$\mathbf{A}^\top$, $\mathbf{A}^{-1}$, $\mathbf{A}^\dagger$	transpose, inverse and pseudoinverse of $\mathbf{A}$
$\lVert\mathbf{A}\rVert$	Frobenius norm of $\mathbf{A}$
$\mathbf{A}^{(n)}$	the n-the matrix of a sequence of matrices
$\odot$	Khatri-Rao product
$\circledast$	Hadamard product
$\oslash$	element-wise division
$\bigodot_{i=1}^{N}\mathbf{A}^{(i)}$	$\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(i)} \odot \cdots \odot \mathbf{A}^{(1)}$
$\circledast_{i=1}^{N}\mathbf{A}^{(i)}$	$\mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(i)} \circledast \cdots \circledast \mathbf{A}^{(1)}$
$\mathbf{X}_{(n)}$	mode-n unfolding of $\mathcal{X}$
$[\![\bullet]\!]$	CP decomposition operator
R	number of components
N	order of tensor
I_n	length of the n-th mode of $\mathcal{X}$
S, J	$\prod_{n=1}^{N-1} I_n$, $\sum_{n=1}^{N-1} I_n$
t, Δt	length of time of existing and incoming tensors
$\lvert\Omega^+\rvert$, $\lvert\Delta\Omega^+\rvert$	number of non-zeros in full and incoming tensors

III. Preliminaries

The notation used through this paper is summarized in Table I. Basically, given an N^{th}-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, its CP decomposition is denoted by $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$. The mode-$n$ unfolding of $\mathcal{X}$ can be approximated as

$$\begin{aligned}\mathbf{X}_{(n)} &\approx \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)})^\top \\ &= \mathbf{A}^{(n)}(\textstyle\bigodot_{i\neq n}^{N} \mathbf{A}^{(i)})^\top.\end{aligned}$$

The n-th matrix $\mathbf{A}^{(n)}$ is updated by ALS as

$$\begin{aligned}\mathbf{A}^{(n)} &\leftarrow \underset{\mathbf{A}^{(n)}}{\arg\min} \frac{1}{2} \left\lVert \mathbf{X}_{(n)} - \mathbf{A}^{(n)}(\odot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top \right\rVert^2 \\ &= \frac{\mathbf{X}_{(n)}(\odot_{i\neq n}^{N}\mathbf{A}^{(i)})}{\circledast_{i\neq 1}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})}.\end{aligned} \tag{1}$$

The major computational bottleneck is the calculation of $\mathbf{X}_{(n)}(\odot_{i\neq n}^{N}\mathbf{A}^{(i)})$, which is often referred to as *matricized tensor times Khatri-Rao products* (MTTKRP) [12]. Specifically, the first term, $\mathbf{X}_{(n)}$, is a $I_n \times \prod_{i\neq n}^{N} I_i$ flat matrix and the Khatri-Rao products produce a super tall matrix with size $\prod_{i\neq n}^{N} I_i \times R$, resulting in the linear time complexity w.r.t. the size of $\mathcal{X}$. In addition, in terms of space usage, ALS is challenged by the so-called intermediate data explosion problem [8], since it requires allocating a huge amount of temporal memory space to store intermediate results.

Overall, such high cost makes typical ALS impractical for large-scale sparse tensors. To address this issue, a number of methods have been proposed by taking sparsity into consideration and using techniques such as distributed and parallel computing [7], [8], [12], [13], and GPU acceleration [14]. The most popular method that has been widely recognized as a gold standard is [11] (summarized in Algorithm 1), which dramatically reduces the complexity of MTTKRP to be linear in the number of non-zeros in a sparse tensor.

Algorithm 1: MTTKRP via Sparse Tensor-Vector Products

```
Input: non-zeros $\mathbf{z}$, indices of non-zeros $\mathbf{j}^{(1)}, \dots, \mathbf{j}^{(N)}$,
       loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$, mode $n$
Output: mode-$n$ MTTKRP $\mathbf{P}^{(n)}$
for $r \leftarrow 1$ to $R$ do
    $\mathbf{t} \leftarrow \mathbf{z}$
    for $i \leftarrow 1$ to $N$ do
        $\mathbf{t} \leftarrow \mathbf{t} \circledast \mathbf{a}^{(i)}_{\mathbf{j}^{(i)}r}$
    end
    $\mathbf{p}^{(n)}_{\cdot r} \leftarrow$ ACCUMARRAY$(\mathbf{j}^{(n)}, \mathbf{t})$
end
```

Algorithm 2: Initialization Procedure of OnlineSCP

```
Input: initial decomposition $[\![\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}]\!]$
Output: helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$
$\mathbf{U}^{(N)} \leftarrow \mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$
$\mathbf{Q} \leftarrow \mathbf{U}^{(N)}$
for $n \leftarrow 1$ to $N-1$ do
    $\mathbf{Q} \leftarrow \mathbf{Q} \circledast (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$
end
```

Algorithm 3: Update Procedure of OnlineSCP

```
Input: loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N-1)}$, helper
       matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$, new data $\Delta\mathcal{X}$
Output: updated loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N-1)}$,
        updated helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$,
        coefficient of new data $\Delta\mathbf{A}^{(N)}$
$\widetilde{\mathbf{Q}} \leftarrow \mathbf{Q}$
// process temporal mode
$\mathbf{Q}^{(N)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(N)}$
$\Delta\mathbf{P}^{(N)} \leftarrow$ MTTKRP by Algorithm 1
$\Delta\mathbf{A}^{(N)} \leftarrow \Delta\mathbf{P}^{(N)}(\mathbf{Q}^{(N)})^{-1}$
// update helper matrices
$\mathbf{U}^{(N)} \leftarrow \mathbf{U}^{(N)} + (\Delta\mathbf{A}^{(N)\top}\Delta\mathbf{A}^{(N)})$
$\mathbf{Q} \leftarrow \mathbf{Q}^{(N)} \circledast \mathbf{U}^{(N)}$
// process non-temporal mode
for $n \leftarrow 1$ to $N-1$ do
    $\widetilde{\mathbf{A}}^{(n)} \leftarrow \mathbf{A}^{(n)}$
    $\mathbf{U}^{(n)} \leftarrow \mathbf{A}^{(n)\top}\mathbf{A}^{(n)}$
    $\mathbf{Q}^{(n)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(n)}$, $\widetilde{\mathbf{Q}}^{(n)} \leftarrow \widetilde{\mathbf{Q}} \oslash \mathbf{U}^{(n)}$
    $\Delta\mathbf{P}^{(n)} \leftarrow$ MTTKRP by Algorithm 1
    $\mathbf{A}^{(n)} \leftarrow (\mathbf{A}^{(n)}\widetilde{\mathbf{Q}}^{(n)} + \Delta\mathbf{P}^{(n)})(\mathbf{Q}^{(n)})^{-1}$
    // update helper matrices
    $\mathbf{Q} \leftarrow \mathbf{Q}^{(n)} \circledast (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$
    $\widetilde{\mathbf{Q}} \leftarrow \widetilde{\mathbf{Q}}^{(n)} \circledast (\widetilde{\mathbf{A}}^{(n)\top}\mathbf{A}^{(n)})$
end
```

IV. Our Approach

This section introduces our approach, OnlineSCP, an algorithm to track the CP decomposition of a sparse online tensor. We first discuss the main idea of our algorithm, followed by key improvements that have been applied and a brief complexity comparison to state-of-the-art methods.

Formally speaking, the research question we solve is defined as follows:

Problem Definition. Given *(i)* an existing CP decomposition $[\![\widetilde{\mathbf{A}}^{(1)}, \dots, \widetilde{\mathbf{A}}^{(N)}]\!]$ of R components that approximates a sparse tensor $\widetilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times t}$ at time t, *(ii)* a new incoming sparse tensor $\Delta\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times \Delta t}$ that is appended to $\widetilde{\mathcal{X}}$ at its last mode and forms a new tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times (t+\Delta t)}$, where $\Delta t \ll t$. The objective is to find the CP decomposition $[\![\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}]\!]$ of $\mathcal{X}$.

A. The Principle of OnlineSCP

To address the online SCP (sparse CP decomposition) problem, our method follows the same alternating update schema as ALS, such that only one loading matrix is updated at one time by fixing all others. Specifically, we update the time mode first, then move on to the rest of the non-temporal modes as

$$\mathbf{A}^{(N)} \to \mathbf{A}^{(1)} \to \mathbf{A}^{(2)} \dots \to \mathbf{A}^{(N-1)}.$$

In order to take advantage of the fact that the tensor is only growing at its time mode, similar to existing online works [9], [10], the main assumption of our method is that the new incoming data, $\Delta\mathcal{X}$, will not have significant impact on the existing model, but only contribute to the update of its corresponding local features. By this, we mean when new data arrives, the first t rows in the loading matrix of the temporal mode, $\mathbf{A}^{(N)}$, will remain unchanged; while the loading matrices for non-temporal modes, $\mathbf{A}^{(1)}, \dots \mathbf{A}^{(N-1)}$, will receive a minor update based on their existing values.

At a high level, as presented in Algorithms 2 and 3, our algorithm consists of two procedures. Before the start of the learning phase, it initializes two small *helper* matrices for storing historical information by using the initial decomposition (details in Algorithm 2). After that, in the online learning phase, the new incoming tensors can be efficiently processed by an incremental update schema (details in Algorithm 3).

Compared to ALS, the major speedup gained by our approach comes from several aspects: (i) only a small fraction of new values are added to the loading matrix of the temporal mode while the remainder is kept unchanged (as discussed in §IV-B); (ii) for non-temporal modes, we break down the costly MTTKRP calculation into historical and new data parts, the historical one is avoided by using helper matrices and the new data one is efficiently obtained by Algorithm 1 (as discussed in §IV-C); (iii) the helper matrices can also be incrementally updated at a small cost (as discussed in §IV-D).

B. Updating the Temporal Mode

Specifically, the temporal loading matrix, $\mathbf{A}^{(N)}$, is a $(t+\Delta t)\times R$ matrix as $[\widetilde{\mathbf{A}}^{(N)};\Delta\mathbf{A}^{(N)}]$, where $\widetilde{\mathbf{A}}^{(N)}$ contains the first t rows of the previous temporal loading matrix. $\Delta\mathbf{A}^{(N)}$ can be easily obtained by projecting $\Delta\mathcal{X}$ to the last mode via other non-temporal loading matrices as

$$\Delta\mathbf{A}^{(N)} = \Delta\mathbf{X}_{(N)}((\odot_{i=1}^{N-1}\mathbf{A}^{(i)})^\top)^\dagger = \frac{\Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})}{\circledast_{i=1}^{N-1}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})}.$$

Let $\mathbf{U}^{(N)} = \mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$, $\mathbf{Q} = \circledast_{i=1}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})$, and $\Delta\mathbf{P}^{(N)} = \Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})$, recall that there is no loading matrices has been updated so far, we can rewrite the above equation with helper matrices $\mathbf{U}^{(N)}$ and $\mathbf{Q}$ as

$$\Delta\mathbf{A}^{(N)} \leftarrow \frac{\Delta\mathbf{P}^{(N)}}{\mathbf{Q}\oslash\mathbf{U}^{(N)}}, \tag{2}$$

where the MTTKRP, $\Delta\mathbf{P}^{(N)}$, can be efficiently calculated by Algorithm 1 at linear complexity to the number of non-zeros in $\Delta\mathcal{X}$, which we refer to as $|\Delta\Omega^+|$.

C. Updating Non-Temporal Modes

Without loss of generality, here we show how to derive the incremental update rule for loading matrix at the first mode, $\mathbf{A}^{(1)}$. By using helper matrix $\mathbf{Q}$ defined as above, the update given by the typical ALS is

$$\mathbf{A}^{(1)} \leftarrow \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\circledast_{i=2}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})} = \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}. \tag{3}$$

Even though the MTTKRP operation, $\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})$, can be accelerated by Algorithm 1 given that $\mathcal{X}$ is sparse, such cost is still not accepatable since $\mathcal{X}$ is potentially a large-scale tensor and the number of non-zeros in $\mathcal{X}$ will keep growing with the increase of time. In fact, by noticing that $\mathcal{X}$ is an online tensor that new data is only appended at its last mode, some patterns can be found in its mode-n unfolding and the corresponding Khatri-Rao product. As a result, we make use of such patterns to derive an efficient update rule as follows.

Recall that $\mathbf{A}^{(N)}$ has been updated as $[\widetilde{\mathbf{A}}^{(N)};\Delta\mathbf{A}^{(N)}]$ and let $\mathbf{B}^{(1)} = \odot_{i=2}^{N-1}\mathbf{A}^{(i)}$, Eq. (3) can be rewritten as

$$\begin{aligned}\mathbf{A}^{(1)} &\leftarrow \frac{\left[\widetilde{\mathbf{X}}_{(1)},\Delta\mathbf{X}_{(1)}\right]\left(\begin{bmatrix}\widetilde{\mathbf{A}}^{(N)}\\ \Delta\mathbf{A}^{(N)}\end{bmatrix}\odot\mathbf{B}^{(1)}\right)}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}\\ &= \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)}) + \Delta\mathbf{X}_{(1)}(\Delta\mathbf{A}^{(N)}\odot\mathbf{B}^{(1)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}\\ &= \frac{\widetilde{\mathbf{P}}^{(1)}+\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}.\end{aligned}$$

In this way, the MTTKRP is divided into two parts: the historical part, $\widetilde{\mathbf{P}}^{(1)}$; and the new data part, $\Delta\mathbf{P}^{(1)}$. Additionally, as $[\![\widetilde{\mathbf{A}}^{(1)},\ldots,\widetilde{\mathbf{A}}^{(N)}]\!]$ is the existing decomposition, we have

$$\widetilde{\mathbf{X}}_{(1)} \approx \widetilde{\mathbf{A}}^{(1)}(\odot_{i=2}^{N}\widetilde{\mathbf{A}}^{(i)})^\top = \widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\widetilde{\mathbf{B}}^{(1)})^\top, \tag{4}$$

where $\widetilde{\mathbf{B}}^{(1)} = \odot_{i=2}^{N-1}\widetilde{\mathbf{A}}^{(i)}$. Based on this, $\widetilde{\mathbf{X}}_{(1)}$ in $\widetilde{\mathbf{P}}^{(1)}$ can be replaced by Eq. (4) and the final update rule for $\mathbf{A}^{(1)}$ is

$$\begin{aligned}\mathbf{A}^{(1)} &\leftarrow \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)})+\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}\\ &\approx \frac{\widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\widetilde{\mathbf{B}}^{(1)})^\top(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)})+\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}\\ &= \frac{\widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)})\circledast(\widetilde{\mathbf{B}}^{(1)\top}\mathbf{B}^{(1)})+\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}\\ &= \widetilde{\mathbf{A}}^{(1)}\frac{\widetilde{\mathbf{Q}}\oslash\mathbf{U}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} + \frac{\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}},\end{aligned} \tag{5}$$

where $\widetilde{\mathbf{Q}} = (\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)})\circledast(\circledast_{i=1}^{N-1}(\widetilde{\mathbf{A}}^{(i)\top}\mathbf{A}^{(i)}))$.

Overall, the above update rule significantly reduces the computational cost by limiting the expensive MTTKRP operation to the new data only. We can interpret such update schema as follows: *the new loading matrix is a weighted combination of existing loading and a hyper loading learned from the new data. The weight between them is determined by the ratio of information contains in the historical data w.r.t. the full data.*

D. Incremental Update of $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$

As mentioned before, by definition we have

$$\mathbf{Q} = (\mathbf{A}^{(1)\top}\mathbf{A}^{(1)})\cdots\circledast(\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})\circledast\cdots(\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}),$$
$$\widetilde{\mathbf{Q}} = (\widetilde{\mathbf{A}}^{(1)\top}\mathbf{A}^{(1)})\cdots\circledast(\widetilde{\mathbf{A}}^{(n)\top}\mathbf{A}^{(n)})\circledast\cdots(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}).$$

The values in both of them are gradually changing with the updating of loading matrices. However, the main difference between the $\mathbf{Q}$ values before and after updating $\mathbf{A}^{(n)}$ is just the $\mathbf{A}^{(n)\top}\mathbf{A}^{(n)}$ term. As a result, we do not need to calculate the $\mathbf{Q}$ values from scratch for each update. Instead, we initialize both $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ as $\circledast_{i=1}^{N}(\widetilde{\mathbf{A}}^{(i)\top}\widetilde{\mathbf{A}}^{(i)})$. After processing the time mode, $\widetilde{\mathbf{Q}}$ remains unchanged, while $\mathbf{Q}$ can be updated as

$$\mathbf{Q}\leftarrow\mathbf{Q}\oslash(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)})\circledast(\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}).$$

Similarly, the updates after the n-th non-temporal mode are

$$\mathbf{Q}\leftarrow\mathbf{Q}\oslash(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)})\circledast(\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}),$$
$$\widetilde{\mathbf{Q}}\leftarrow\widetilde{\mathbf{Q}}\oslash(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)})\circledast(\widetilde{\mathbf{A}}^{(N)\top}\mathbf{A}^{(N)}).$$

After processing all modes, the final $\mathbf{Q}$ is stored and used to initialize $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ for the next batch of new data.

Additionally, since $\mathcal{X}$ is an online tensor and the length of the time mode t can be potentially quite large, we choose to avoid directly computing $\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}$ and $\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$ by storing $\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}$ into a small $R\times R$ matrix $\mathbf{U}^{(N)}$, and $\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$ can be easily obtained as

$$\mathbf{U}^{(N)}\leftarrow\mathbf{U}^{(N)}+(\Delta\mathbf{A}^{(N)\top}\Delta\mathbf{A}^{(N)}).$$

To sum up, our proposed algorithm, OnlineSCP, is consisted of two procedures: it first initialize two small helper matrices with the initial tensor and its decomposition (presented in Algorithm 2), then the new incoming tensors can be efficiently processed by our efficient incremental update schema (presented in Algorithm 3). A complexity comparison to the state-of-the-arts can be found in Table II.

TABLE II
COMPLEXITY COMPARISON

	Time	Memory
OnlineSCP	$(J+\Delta t)R^2 + \lvert\Delta\Omega^+\rvert NR$	$(J+\Delta t)R + \lvert\Delta\Omega^+\rvert$
ALS	$(J+t+\Delta t)R^2 + \lvert\Omega^+\rvert NR$	$(J+t+\Delta t)R + \lvert\Omega^+\rvert$
OnlineCP	$(J+\Delta t)R^2 + NRS\Delta t$	$(J+\Delta t)R + SR\Delta t$
SDT	$(S+\Delta t)R^2$	$(J+S+t)R + SR\Delta t$
RLST	SR^2	$(S+J+\Delta t)R + SR\Delta t$

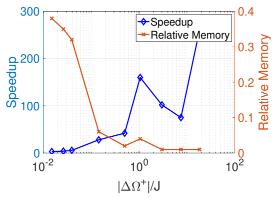

Fig. 2. The efficiency of OnlineSCP is correlated to the ratio between the number of non-zeros in data and the overall length of its non-temporal modes

V. Empirical Analysis

This section compares the performance of our proposed OnlineSCP algorithm with state-of-the-art techniques, measured by their effectiveness and efficiency on both time and space.

A. Experiment Specifications

Nine real-world datasets (details in Table III) and four baselines, including ALS [21], OnlineCP [10], SDT and RLST [9], have been used in our experiments.

The performance is evaluated by effectiveness and efficiency. In terms of effectiveness, we measure the *fitness* of each algorithm as $1 - \frac{\|\hat{\mathcal{X}} - \mathcal{X}\|}{\|\mathcal{X}\|}$, where $\mathcal{X}$ is the original data and $\hat{\mathcal{X}}$ is the estimation. For efficiency, both the *running time* and *memory usage* for processing one batch of new data are measured. Since ALS is the most popular method for CP decomposition, we report the *relative performance* of an online method to ALS over all three metrics.

Given a dataset, the first 50% data along the last mode is decomposed by ALS and used for initializing all methods. After that, the second half data is further divided into 100 batches and each of them is sequentially appended to the existing data. All methods process one batch of data at a time. After learning a new batch, the updated decompositions of all methods are used to calculate the fitness, along with the time and memory consumption for this batch.

For each algorithm-dataset pair, the experiment is conducted with 4 CPU cores, 32 GB memory and 12-hour maximum running time. The whole experiment is replicated 5 times and the final results are averaged over these repetitions. For reproducibility, the Matlab implementation of our OnlineSCP algorithm and a more detailed experiment specification can be found at http://shuo-zhou.info.

B. Results and Discussions

The experimental results are shown in Table IV, V and VI. Among all online methods, our proposal, OnlineSCP, is the only method that is able to produce results for all datasets, given the limited computational resources as stated before. In contrast, SDT and RLST only manage to process MovieLens and LastFM, which are two relatively small datasets in terms of slice size (6K $\times$ 6K for MovieLens and 1K $\times$ 1K for LastFM). OnlineCP can process one more dataset compared to them, NELL-2, which has a slice size of 108M. Ideally OnlineCP should also work on NIPS dataset as one slice of NIPS data is slightly smaller than that of NELL-2. However, OnlineCP has a specifically designed procedure to speedup calculations on higher-order tensors by costing more memory, hence it fails on this 4^{th}-order tensor.

We can see that the efficiency of our OnlineSCP method varies from dataset to dataset. For example, compared to ALS, the speedups of OnlineSCP on the social network datasets, Facebook-Links, Facebook-Wall and Youtube are only 3.9, 5.65 and 3.14 times, respectively; while it can improve the time consumption on NIPS, Enron and NELL-2 by more than 100 times. Similar patterns can be found for memory usage as well, where for social network data, around one third of memory used by ALS is needed for OnlineSCP, while for others the relative memory can be as less as 1% w.r.t. ALS.

One can understand this behavior based on the complexity analysis. Specifically, the complexity of OnlineSCP, both in time and space, consists of two parts: 1) one part related to the overall length of the non-temporal modes, J, 2) and the other part that depends on the number of non-zeros in the new data, $|\Delta\Omega^+|$. That is, if J is a relatively large number compared to $|\Delta\Omega^+|$, then the speedup obtained by the incremental learning phase will be significantly exploited. In fact, by plotting $|\Delta\Omega^+|/J$ against the speedup (or relative memory) of our method w.r.t. ALS, we can see a clear correlation between this ratio and the efficiency as shown in Figure 2. This means that our method tends to work better for tensors that have relatively smaller overall length of the non-temporal modes. However, this does not necessarily means that our method is not suitable for large-scale sparse online tensors. In fact, it should be highlighted that in this experiment setting, each new data batch contains around 0.5% of overall data, so the maximum speedup can be gained in the MTTKRP calculation is about 200 times, compared to the ALS algorithm. In practice, as long as the time used by our algorithm for one new batch of data is less than the data generation time, one can always use a smaller batch size to gain even greater efficiency.

VI. Conclusions

To conclude, in this paper we proposed an efficient algorithm, OnlineSCP, to incrementally track the up-to-date CP decomposition when a new batch of data is appended to the time mode of a sparse tensor. The complexity of our algorithm is only linear to the number of non-zeros in the new data, which is significantly lower than existing online approaches that are designed for dense tensors. As evaluated on nine real-world datasets, our algorithm demonstrated considerable improvements over state-of-the-art techniques, in terms of decomposition quality, time and space efficiency.

TABLE III
DETAIL OF REAL-WORLD DATASETS (K: THOUSANDS, M: MILLIONS)

Datasets	Description	Size	nnz*	Density	Batch Size
Facebook-Links	user × user × day	64K × 64K × 886	671K	2×10^{-7}	4
Facebook-Wall	wall owner × poster × day	47K × 47K × 2K	738K	2×10^{-7}	8
MovieLens	user × movie × time	6K × 4K × 1K	1M	4×10^{-5}	5
LastFM	user × artist × time	1K × 1K × 168	3M	2×10^{-2}	1
NIPS	paper × author × year × word	2K × 3K × 17 × 14K	3M	2×10^{-6}	70
Youtube	user × user × day	3M × 3M × 226	18M	8×10^{-9}	1
Enron	sender × receiver × word × day	6K × 6K × 244K × 1K	54M	5×10^{-9}	6
NELL-2	entity × relation × entity	12K × 9K × 30K	77M	2×10^{-5}	144
NELL-1	entity × relation × entity	3M × 2M × 25M	144M	9×10^{-13}	127,476

* number of non-zeros

TABLE IV
MEAN RELATIVE **FITNESS** TO ALS OVER ALL BATCHES.
THE HIGHER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a*	n/a	n/a	**1.00**
Facebook-Wall	n/a	n/a	n/a	**0.92**
MovieLens	**1.00**	0.31	**1.00**	**1.00**
LastFM	**0.91**	0.22	0.17	**0.91**
NIPS	n/a	n/a	n/a	**0.96**
Youtube	n/a	n/a	n/a	**1.00**
Enron	n/a	n/a	n/a	**0.93**
NELL-2	0.97	n/a	n/a	**0.98**
NELL-1	n/a	n/a	n/a	**0.84**

* n/a means the method is failed since it is not applicable/running out of memory/cannot finish within 12 hours

TABLE V
MEAN RELATIVE **SPEEDUP** TO ALS OVER ALL BATCHES.
THE HIGHER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a	n/a	n/a	**3.90**
Facebook-Wall	n/a	n/a	n/a	**5.65**
MovieLens	2.03	0.05	0.02	**42.06**
LastFM	70.68	12.41	6.53	**74.83**
NIPS	n/a	n/a	n/a	**102.08**
Youtube	n/a	n/a	n/a	**3.14**
Enron	n/a	n/a	n/a	**160.24**
NELL-2	30.12	n/a	n/a	**258.50**
NELL-1	n/a	n/a	n/a	**27.81**

TABLE VI
MEAN RELATIVE **MEMORY USAGE** TO ALS OVER ALL BATCHES.
THE LOWER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a	n/a	n/a	**0.35**
Facebook-Wall	n/a	n/a	n/a	**0.32**
MovieLens	17.41	60.87	46.38	**0.02**
LastFM	0.36	1.24	0.94	**0.01**
NIPS	n/a	n/a	n/a	**0.01**
Youtube	n/a	n/a	n/a	**0.38**
Enron	n/a	n/a	n/a	**0.04**
NELL-2	1.49	n/a	n/a	**0.01**
NELL-1	n/a	n/a	n/a	**0.06**

REFERENCES

[1] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[2] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[3] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, "Euclidean Embedding of Co-occurrence Data," *JMLR*, vol. 8, pp. 2265–2295, 2007.

[4] T. G. Kolda, B. W. Bader, and J. P. Kenny, "Higher-order web link analysis using multilinear algebra," in *ICDM*, 2005.

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

[6] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *NIPS*, 2014.

[7] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *ICDE*, 2015.

[8] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *SIGKDD*, 2012.

[9] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," *IEEE Transactions on Signal Processing*, vol. 57, no. 6, pp. 2299–2310, 2009.

[10] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online cp decompositions for higher order tensors," in *SIGKDD*, 2016.

[11] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.

[12] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *IPDPS*, 2015.

[13] J. Li, J. Choi, I. Perros, J. Sun, and R. Vuduc, "Model-driven sparse cp decomposition for higher-order tensors," in *IPDPS*, 2017.

[14] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi, "A unified optimization approach for sparse tensor operations on gpus," in *CLUSTER*, 2017.

[15] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," in *SIGKDD*, 2006.

[16] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM TKDD*, vol. 2, no. 3, p. 11, 2008.

[17] E. Gujral, R. Pasricha, and E. E. Papalexakis, "Sambaten: Sampling-based batch incremental tensor decomposition," in *SDM*, 2018.

[18] S. Zhou, S. M. Erfani, and J. Bailey, "Sced: A general framework for sparse tensor decomposition with constraints and elementwise dynamic learning," in *ICDM*, 2017.

[19] K. Shin, B. Hooi, J. Kim, and C. Faloutsos, "Densealert: Incremental dense-subtensor detection in tensor streams," in *SIGKDD*, 2017.

[20] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in *SIGKDD*, 2017.

[21] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online, February 2015