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

Mining Billion-Scale Tensors: Algorithms and Discoveries

Inah Jeon · Evangelos E. Papalexakis · Christos Faloutsos · Lee Sael · U Kang

Received: date / Accepted: date

Abstract How can we analyze large-scale real-world data with various attributes? Many real-world data (e.g., network traffic logs, web data, social networks, knowledge bases, and sensor streams) with multiple attributes are represented as multi-dimensional arrays, called tensors. For analyzing a tensor, tensor decompositions are widely used in many data mining applications: detecting malicious attackers in network traffic logs with (source IP, destination IP, port-number, timestamp), finding telemarketers in a phone call history with (sender, receiver, date), and identifying interesting concepts in a knowledge base with (subject, object, relation). However, current tensor decomposition methods do not scale to large and sparse real-world tensors with millions of rows and columns and 'fibers'. In this paper, we propose HATEN2, a distributed method for large-scale tensor decompositions that runs on the MAPREDUCE framework. Our careful design and implementation of HATEN2 dramatically reduce the size of intermediate data, and the number of jobs leading to achieving high scalability compared with the state-of-the-art method. Thanks to HATEN2, we analyze big real-world sparse tensors that can not be handled by the current state of the art, and discover hidden concepts.

Keywords Tensor · Distributed Computing · Big Data · MapReduce · Hadoop

Inah Jeon
LG Electronics
E-mail: june5324@gmail.com

Evangelos E. Papalexakis
Computer Science Department & iLab, CMU, U.S.
E-mail: epapalex@cs.cmu.edu

Christos Faloutsos
Computer Science Department & iLab, CMU, U.S.
E-mail: christos@cs.cmu.edu

Lee Sael
Department of Computer Science, SUNY Korea
E-mail: sael@sunykorea.ac.kr

U Kang*
Department of Computer Science and Engineering, Seoul National University, Korea
E-mail: ukang@snu.ac.kr
* Corresponding author

1 Introduction

Given historic records of millions of people calling each other, how can we identify telemarketers who make a lot of calls but never receive ones? How can we detect suspicious attackers in a network packet transmission log? In general, how can we analyze large-scale real-world data with various attributes? Many real-world data (e.g., knowledge bases [1], web data [2], network traffic data [3], and many others [4–6]) with multiple attributes are represented as multi-dimensional arrays, called tensors. In analyzing a tensor, tensor decompositions are powerful tools in many data mining applications: correlation analysis on sensor streams [4], latent semantic indexing on DBLP publication data [7], multi-aspect forensics on network data [3], network discovery on fMRI data [6], to name a few. Using tensor decompositions, we find latent factors (or relations) within the data. These latent factors can be roughly and informally seen as soft-clustering of the data. For example decomposing a tensor constructed from the phone call history with (sender, receiver, date) into R latent factors corresponds to finding R clusters of senders that call a set of receivers on a specific date.

PARAFAC and Tucker are two widely used tensor decompositions. Since there is no single generalization of Singular Value Decomposition (SVD) for a tensor, both PARAFAC and Tucker are considered as extensions of SVD to higher dimensions. PARAFAC is useful for decomposing a tensor into rank-one tensors which form the latent factors;

Tucker is more suitable for compressing tensors and examining relations between the latent factors. Tucker is a more generalized version of PARAFAC, since the factors interact with all pairs of other factors.

Tensor decompositions have been extensively studied for various data mining tasks. However, most of tensor decomposition algorithms designed to deal with tensors fitting in main memory fail to decompose recent large-scale real world tensors with millions and billions of rows, columns, and 'fibers'. The main challenge is to overcome the intermediate data explosion problem in distributed systems where the size of intermediate data exceeds the capacity of a single machine or even a cluster as the size of an input tensor gets larger. For example, if the size of the input tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ is $I = J = K = 1$ millions, the size of intermediate data of the n-mode product in Tucker would become 1 Exabytes (=10^{18} bytes) with a straightforward implementation using distributed systems. Thus, we need to develop scalable and distributed tensor decomposition algorithms.

In this paper, we propose HATEN2 (which stands for HADOOP Tensor method for 2 decompositions), a scalable tensor decomposition suite for Tucker and PARAFAC decompositions on HADOOP [8], an open-source version of the MAPREDUCE framework [9]. HATEN2 provides both unconstrained and nonneagtivity-constrained PARAFAC and Tucker decompositions: HATEN2-PARAFAC and HATEN2-Tucker for unconstrained version, and HATEN2-PARAFACNN and HATEN2-TuckerNN for nonnegativity-constrained version. Nonnegative tensor decomposition receives increasing attention because its result gives the benefit of easy interpretation, thanks to the nonnegativity in the factors. By carefully reordering operations and exploiting the sparsity of real world tensors, HATEN2 solves the intermediate data explosion problem in distributed systems. Furthermore, HATEN2 significantly reduces the running time by integrating several redundant jobs. As a result, HATEN2 is able to analyze data that are several orders of magnitude larger than what the state of the art can handle. Applying HATEN2 to several real world tensors, we discover malicious attackers in network traffic logs with (source IP, destination IP, port-number, timestamp), telemarketers in a phone call history with (sender, receiver, date), and interesting concepts in a knowledge base with (subject, object, relation).

Our main contributions are the following:

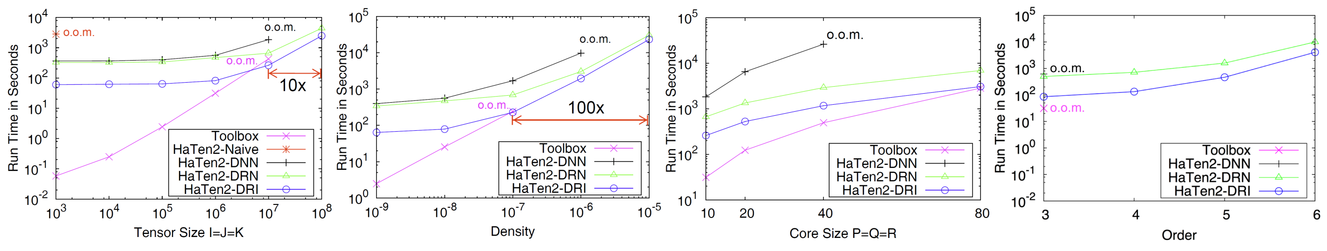
- **Algorithm.** HATEN2 provides a unified framework, for unconstrained and nonnegativity-constrained Tucker and PARAFAC decompositions for sparse real world tensors in distributed systems, which significantly reduce the intermediate data size and the running time.
- **Scalability.** HATEN2 decomposes up to $100\times$ larger tensors without constraints as shown in Figure 1 and up to $1000\times$ larger tensors with the nonnegativity constraint as shown in Figure 8. Furthermore, HATEN2 scales up near linearly on the number of machines.
- **Discovery.** By applying HATEN2, we discover interesting patterns on various real-world data—knowledge bases, network traffic logs, and phone call history—with millions of rows, columns, and fibers which were hard to analyze by existing methods.

Table 1: Table of symbols.

Symbol	**Definition**
$\mathcal{X}$	a tensor
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor
a	a scalar (lowercase, italic letter)
$\mathbf{a}$	a column vector (lowercase, bold letter)
$\mathbf{A}$	a matrix (uppercase, bold letter)
R	number of components
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product
$*$	Hadamard product
$\cdot$	standard product
$\bar{\times}_n$	n-mode vector product
$\times_n$	n-mode matrix product
$\bar{*}_n$	n-mode vector Hadarmard product (Definition 1)
$*_n$	n-mode matrix Hadarmard product (Definition 5)
$\mathbf{A}^T$	transpose of $\mathbf{A}$
$\|\|\mathbf{M}\|\|_F$	Frobenius norm of $\mathbf{M}$
$bin(\mathcal{X})$	function that converts non-zero elements of $\mathcal{X}$ to 1
$nnz(\mathcal{X})$	number of nonzero elements in $\mathcal{X}$
$idx(\mathcal{X})$	set of indices ((i, j, k) or (i, j, k, l)) of nonzero elements in $\mathcal{X}$
I, J, K	dimensions of each mode of input tensor $\mathcal{X}$
P, Q, R	dimensions of each mode of core tensor $\mathcal{G}$

The binary code and datasets are available at `http://datalab.snu.ac.kr/haten2`. The preliminary version of this work is described in [10]. In this work, we add two tensor decompositions (HaTen2-PARAFACNN and HaTen2-TuckerNN) for nonnegativity-constraints, and formulate them using our HaTen2 framework (Section 3.3). Furthermore, we present the additional discovery results on the three real world tensors: concept discovery results on RDF knowledge base (Freebase-sampled), network traffic pattern discovery results on network traffic logs, and phone call pattern discovery results on phone call history (Phonecall) (Section 5.1$\sim$ 5.3).

The rest of paper is organized as follows. Section 2 presents the preliminaries of the tensor and its decompositions. Section 3 describes our proposed method for the scalable tensor decompositions. After presenting the experimental results and discoveries in Section 4 and Section 5, we discuss related works in Section 6. Then we conclude in Section 7.



(a) Nonzeros and Dimensionality (b) Density (c) Core (d) Order

Fig. 1: Data scalability of our proposed HATEN2-DRI compared to other methods, for Tucker decomposition. The datasets are explained in detail at Section 4.1. o.o.m.: out of memory. Density is defined to be the number of nonzeros divided by the number of all elements in a tensor. Note that our best method HATEN2-DRI analyzes $10 \sim 100\times$ larger data than the Tensor Toolbox.

2 Preliminaries

In this section, we describe the preliminaries on tensor and its decompositions. Table 1 shows the definitions of symbols used in this paper. Matrices are denoted by boldface capitals (e.g. $\mathbf{B}$), and the rth row of the matrix $\mathbf{B}$ is denoted by $\mathbf{b}_r$. Vectors are denoted by boldface lowercases (e.g. $\mathbf{a}$).

2.1 Tensor

Tensor. Tensor is a multi-dimensional array. Each 'dimension' of a tensor is called *mode* or *way*. An N-mode or N-way tensor is denoted by $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$. $bin(\mathcal{X})$ denotes a function that converts non-zero elements in $\mathcal{X}$ to 1. $nnz(\mathcal{X})$ means the number of non-zero elements of $\mathcal{X}$, and $idx(\mathcal{X})$ means the set of indexes (e.g. (i,j,k) for 3-mode tensor $\mathcal{X}$) of non-zero elements in $\mathcal{X}$.
Fibers and Slices. A fiber is defined by fixing all but one index. In a 3-way tensor, it is denoted by $\mathcal{X}_{:jk}, \mathcal{X}_{i:k}$, and $\mathcal{X}_{ij:}$. A slice is defined by fixing all indices but two indices. In a 3-way tensor, it is denoted by $\mathcal{X}_{i::}, \mathcal{X}_{:j:}$ and $\mathcal{X}_{::k}$.
Matricization of tensor. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ is denoted by $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n\times(\prod_{k\neq n} I_k)}$ and arranges the mode-n fibers to be the columns of the resulting matrix.
n-mode matrix product. The n-mode matrix product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J\times I_n}$ is denoted by $\mathcal{X}\times_n\mathbf{U}$ and is of size $I_1\times\cdots I_{n-1}\times J\times I_{n+1}\cdots\times I_N$. It is defined by

$$(\mathcal{X}\times_n \mathbf{U})_{i_1\ldots i_{n-1}ji_{n+1}\ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1i_2\ldots i_N}u_{ji_n}.$$

n-mode vector product. The n-mode vector product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ with a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X}\bar{\times}_n\mathbf{v}$ and is of size $I_1\times\cdots I_{n-1}\times I_{n+1}\cdots\times I_N$. It is defined by

$$(\mathcal{X}\bar{\times}_n\mathbf{v})_{i_1\ldots i_{n-1}i_{n+1}\ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1i_2\ldots i_N}v_{i_n}.$$

Kronecker product. The Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{K\times L}$ is denoted by $\mathbf{A}\otimes\mathbf{B}$. The result is a matrix of size $(IK)\times(JL)$ and defined by

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} \mathbf{a}_{11}\mathbf{B} & \mathbf{a}_{12}\mathbf{B} & \cdots & \mathbf{a}_{1J}\mathbf{B} \\ \mathbf{a}_{21}\mathbf{B} & \mathbf{a}_{22}\mathbf{B} & \cdots & \mathbf{a}_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{I1}\mathbf{B} & \mathbf{a}_{I2}\mathbf{B} & \cdots & \mathbf{a}_{IJ}\mathbf{B} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{a}_1\otimes\mathbf{b}_1 & \mathbf{a}_1\otimes\mathbf{b}_2 & \mathbf{a}_1\otimes\mathbf{b}_3 & \cdots & \mathbf{a}_J\otimes\mathbf{b}_{L-1} & \mathbf{a}_J\otimes\mathbf{b}_L \end{bmatrix}$$

Khatri-Rao product. The Khatri-Rao product (or column-wise Kronecker product) $(\mathbf{A}\odot\mathbf{B})$, where $\mathbf{A},\mathbf{B}$ have the same number of columns, say R, is defined as:

$$\mathbf{A}\odot\mathbf{B} = \begin{bmatrix} \mathbf{A}(:,1)\otimes\mathbf{B}(:,1) \cdots \mathbf{A}(:,R)\otimes\mathbf{B}(:,R) \end{bmatrix}$$

where $\mathbf{A}(:,r)$ is the rth column of $\mathbf{A}$. If $\mathbf{A}$ is of size $I\times R$ and $\mathbf{B}$ is of size $J\times R$ then $(\mathbf{A}\odot\mathbf{B})$ is of size $IJ\times R$.
Hadamard product. The Hadamard product $\mathbf{A}*\mathbf{B}$ is the elementwise matrix product, where $\mathbf{A}$ and $\mathbf{B}$ have the same size $(I\times J)$, and is defined as:

$$\mathbf{A}*\mathbf{B} = \begin{bmatrix} \mathbf{a}_{11}\mathbf{b}_{11} & \mathbf{a}_{12}\mathbf{b}_{12} & \cdots & \mathbf{a}_{1J}\mathbf{b}_{1J} \\ \mathbf{a}_{21}\mathbf{b}_{21} & \mathbf{a}_{22}\mathbf{b}_{22} & \cdots & \mathbf{a}_{2J}\mathbf{b}_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{I1}\mathbf{b}_{I1} & \mathbf{a}_{I2}\mathbf{b}_{I2} & \cdots & \mathbf{a}_{IJ}\mathbf{b}_{IJ} \end{bmatrix}$$

2.2 Basic Tensor Decomposition

Tensor decomposition is a general tool for tensor analysis. Using tensor decomposition, we find latent factor or relations among data. In this paper, we focus on two major tensor decompositions, PARAFAC and Tucker.

2.2.1 PARAFAC Decomposition

PARAFAC (parallel factors) decomposition [11], also called CANDECOMP (canonical decomposition), decomposes a tensor into a sum of rank-one tensors. There has been rich literature on algorithms for the PARAFAC decomposition, a concise summary thereof can be found in [12].

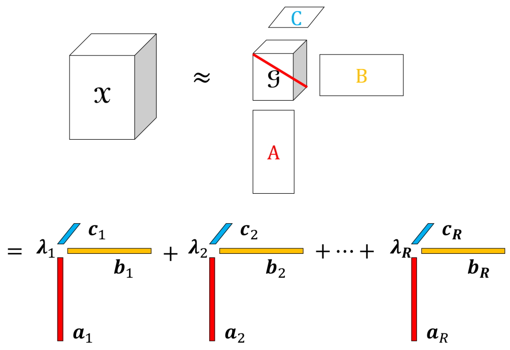


Fig. 2: Rank-R PARAFAC decomposition of a three-way tensor. The tensor $\mathcal{X}$ is decomposed as three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

PARAFAC decomposition for 3-way tensor. Given a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ and rank R, PARAFAC decomposition factorizes the tensor into 3 factor matrices, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, as follows:

$$\mathcal{X} \approx [\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}] = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

where R is a positive integer (typically between 10 and 100), λ is a weight vector, and $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$ are the factor matrices. Figure 2 shows the 3-way PARAFAC tensor decomposition.

PARAFAC decomposition for N-way tensor. Given an N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times \ldots\times I_N}$ and rank R, PARAFAC decomposition factorizes the tensor into N factor matrices, $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, ... , $\mathbf{A}^{(N)}$, as follows:

$$\mathcal{X} \approx [\lambda; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(N)}] = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \ldots \circ \mathbf{a}_r^{(N)}.$$

where R is a positive integer, λ is a weight vector, and $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1\times R}$, $\mathbf{A}^{(2)} \in \mathbb{R}^{I_2\times R}$, ... , $\mathbf{A}^{(N)} \in \mathbb{R}^{I_N\times R}$ are the factor matrices.

PARAFAC-ALS. Algorithm 1 shows the alternating least squares algorithm for 3-way PARAFAC decomposition where † denotes the pseudo-inverse operation. The stopping criterion for Algorithm 1 is either one of the following: 1) the difference between the two least squares errors of consecutive iterations is smaller than a threshold, or 2) the maximum number of iterations is exceeded.

Algorithm 1: 3-way PARAFAC-ALS.

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, rank $R$, maximum iterations $T$
Output: PARAFAC decomposition $\boldsymbol{\lambda} \in \mathbb{R}^{R\times 1}$, $\mathbf{A} \in \mathbb{R}^{I\times R}$,
    $\mathbf{B} \in \mathbb{R}^{J\times R}$, $\mathbf{C} \in \mathbb{R}^{K\times R}$
1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$;
2: for $t = 1, ..., T$ do
3:     $\mathbf{A} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B}) (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^\dagger$;
4:     Normalize columns of $\mathbf{A}$ (storing norms in vector $\boldsymbol{\lambda}$);
5:     $\mathbf{B} \leftarrow \mathbf{X}_{(2)} (\mathbf{C} \odot \mathbf{A}) (\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^\dagger$;
6:     Normalize columns of $\mathbf{B}$ (storing norms in vector $\boldsymbol{\lambda}$);
7:     $\mathbf{C} \leftarrow \mathbf{X}_{(3)} (\mathbf{B} \odot \mathbf{A}) (\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^\dagger$;
8:     Normalize columns of $\mathbf{C}$ (storing norms in vector $\boldsymbol{\lambda}$);
9:     if stopping criterion is met then
10:        break for loop;
11:    end if
12: end for
13: return $\boldsymbol{\lambda}, \mathbf{A}, \mathbf{B}, \mathbf{C}$;
```

2.2.2 Tucker Decomposition

In Tucker decomposition [13], called N-mode PCA or N-mode SVD, a tensor is decomposed into a core tensor and factor matrices of each mode. The factor matrices represent the principal components of each mode and the core tensor represents the interactions between the different components. Tucker decomposition is a more generalized version of PARAFAC decomposition, since the factors interact with all pairs of other factors.

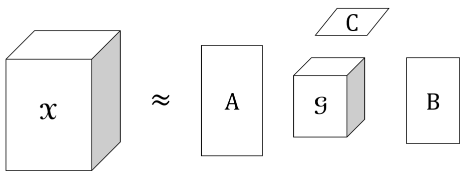


Fig. 3: Tucker decomposition of a three-way tensor. The tensor $\mathcal{X}$ is decomposed as a core tensor $\mathcal{G}$, and three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

Tucker decomposition for 3-way tensor. The 3-way tensor is decomposed as follows:

$$\mathcal{X} \approx [\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}] = \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$$

$$= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r$$

where $\mathcal{G} \in \mathbb{R}^{P\times Q\times R}$ is the core tensor, and $\mathbf{A} \in \mathbb{R}^{I\times P}, \mathbf{B} \in \mathbb{R}^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$ are the factor matrices. Figure 3 shows the Tucker decomposition of a 3-way tensor.

Tucker decomposition for N-way tensor. The N-way tensor is decomposed as follows.

$$\mathcal{X} \approx [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(N)}]$$
$$= \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} ... \times_N \mathbf{A}^{(N)}$$

where, $\mathcal{G} \in \mathbb{R}^{J_1\times J_2 ... \times J_N}$ is the core tensor, and $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1\times J_1}, \mathbf{A}^{(2)} \in \mathbb{R}^{I_2\times J_2}$, ..., and $\mathbf{A}^{(N)} \in \mathbb{R}^{I_N\times J_N}$ are the factor matrices.

Tucker-ALS. Tucker-ALS algorithm uses an alternating least squares approach. For updating each factor, there are some approaches such as SVD, Bauer-Rutishauser, Gram-Schmidt and NIPALS [14]. Algorithm 2 shows the standard SVD-based algorithm for 3-way Tucker decomposition.

Algorithm 2: 3-way Tucker-ALS

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, desired core size: $P \times Q \times R$
Output: Core tensor $\mathcal{G} \in \mathbb{R}^{P\times Q\times R}$ and orthogonal factor matrices
        $\mathbf{A} \in \mathbb{R}^{I\times P}, \mathbf{B} \in \mathbb{R}^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$
 1: Initialize $\mathbf{B}, \mathbf{C}$;
 2: repeat
 3:     $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$;
 4:     $\mathbf{A} \leftarrow P$ leading left singular vectors of $\mathbf{Y}_{(1)}$;
 5:     $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_3 \mathbf{C}^T$;
 6:     $\mathbf{B} \leftarrow Q$ leading left singular vectors of $\mathbf{Y}_{(2)}$;
 7:     $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^T \times_2 \mathbf{B}^T$;
 8:     $\mathbf{C} \leftarrow R$ leading left singular vectors of $\mathbf{Y}_{(3)}$;
 9:     $\mathcal{G} \leftarrow \mathcal{Y} \times_3 \mathbf{C}$;
10: until $||\mathcal{G}||$ ceases to increase or the maximum number of outer
    iterations is exceeded.
```

2.3 Nonnegative Tensor Decomposition

Nonnegative tensor decomposition (NTD) is a tensor decomposition with the constraint enforcing all elements in the factors to be nonnegative. Since all elements have nonnegative values, the result of NTD is more interpretable. The most widely used method for nonnegative tensor decomposition or nonnegative matrix decomposition is the multiplicative update rule proposed by Lee and Seung [15]. They use the Euclidean distance and Kullback-Leibler divergence for the cost function. Lemma 1 [15] is the Euclidean distance version of the multiplicative update rule for nonnegative matrix decomposition.

Lemma 1 *When factoring the input matrix* $\mathbf{V}$ *into* $\mathbf{WH}$, *the Euclidean distance* $||\mathbf{V} - \mathbf{WH}||$ *is nonincreasing under the update rules*

$$\mathbf{H}_{a\mu} \leftarrow \mathbf{H}_{a\mu} \frac{(\mathbf{W}^T\mathbf{V})_{a\mu}}{(\mathbf{W}^T\mathbf{WH})_{a\mu}}, \mathbf{W}_{a\mu} \leftarrow \mathbf{W}_{a\mu} \frac{(\mathbf{V}^T\mathbf{H})_{a\mu}}{(\mathbf{WHH}^T)_{a\mu}}.$$

The Euclidean distance is invariant under these updates if and only if $\mathbf{W}$ *and* $\mathbf{H}$ *are at a stationary point of the distance.*

Since this rule is composed of only element wise multiplication and division, if an initial matrix is nonnegative, then all interim matrices become nonnegative. The convergence of the multiplicative update rule is proved in the paper [15].

2.3.1 Nonnegative PARAFAC Decomposition

The definition and algorithm of nonnegative PARAFAC decomposition for 3-way tensor are as follows.

Nonnegative PARAFAC decomposition for 3-way tensor. Given a nonnegative tensor $\mathcal{X}$, the nonnegative PARAFAC decomposition solves

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} ||\mathcal{X} - \sum_{r=1}^{R} \boldsymbol{\lambda}_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r||,$$

subject to $\mathbf{a}_r \geq 0, \mathbf{b}_r \geq 0, \mathbf{c}_r \geq 0$, where $\mathbf{A} \in \mathbb{R}_+^{I\times R}, \mathbf{B} \in \mathbb{R}_+^{J\times R}$, and $\mathbf{C} \in \mathbb{R}_+^{K\times R}$ are factor matrices of PARAFAC decomposition.

Nonnegative PARAFAC-ALS. Algorithm 3 shows nonnegative PARAFAC-ALS algorithm for a 3-way tensor using multiplicative update rule [16]. The stopping criterion of Algorithm 3 is the same as that of Algorithm 1.

Algorithm 3: 3-way nonnegative PARAFAC-ALS

```
Input: Tensor $\mathcal{X} \in \mathbb{R}_+^{I\times J\times K}$, rank $R$, maximum iterations $T$.
Output: PARAFAC decomposition $\boldsymbol{\lambda} \in \mathbb{R}^{R\times 1}, \mathbf{A} \in \mathbb{R}_+^{I\times R}$,
        $\mathbf{B} \in \mathbb{R}_+^{J\times R}, \mathbf{C} \in \mathbb{R}_+^{K\times R}$.
 1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ with nonnegative values;
 2: for $t = 1, ..., T$ do
 3:     $\mathbf{A} \leftarrow \mathbf{A} * \frac{\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})}{\mathbf{A}(\mathbf{C}^T\mathbf{C}*\mathbf{B}^T\mathbf{B})}$;
 4:     Normalize columns of $\mathbf{A}$ (storing norms in vector $\boldsymbol{\lambda}$);
 5:     $\mathbf{B} \leftarrow \mathbf{B} * \frac{\mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A})}{\mathbf{B}(\mathbf{C}^T\mathbf{C}*\mathbf{A}^T\mathbf{A})}$;
 6:     Normalize columns of $\mathbf{B}$ (storing norms in vector $\boldsymbol{\lambda}$);
 7:     $\mathbf{C} \leftarrow \mathbf{C} * \frac{\mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A})}{\mathbf{C}(\mathbf{B}^T\mathbf{B}*\mathbf{A}^T\mathbf{A})}$;
 8:     Normalize columns of $\mathbf{C}$ (storing norms in vector $\boldsymbol{\lambda}$);
 9:     if stopping criterion is met then
10:         break for loop;
11:     end if
12: end for
13: return $\boldsymbol{\lambda}, \mathbf{A}, \mathbf{B}, \mathbf{C}$;
```

2.3.2 Nonnegative Tucker Decomposition

The definition and algorithm of the nonnegative Tucker decomposition for 3-way tensor are as follows.

Nonnegative Tucker decomposition for 3-way tensor. Given a nonnegative tensor $\mathcal{X}$, the nonnegative Tucker decomposition solves

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|\mathcal{X} - \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}\mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r\|,$$

subject to $g_{pqr} \geq 0, \mathbf{a}_p \geq 0, \mathbf{b}_q \geq 0, \mathbf{c}_r \geq 0$, where $\mathcal{G} \in \mathbb{R}_+^{P\times Q\times R}$ is the core tensor, and $\mathbf{A} \in \mathbb{R}_+^{I\times P}, \mathbf{B} \in \mathbb{R}_+^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}_+^{K\times R}$ are factor matrices of Tucker decomposition.

Nonnegative Tucker-ALS. Kim and Choi extend multiplicative rule based nonnegative PARAFAC decomposition to nonnegative Tucker decomposition. Algorithm 4 shows nonnegative Tucker-ALS algorithm for a 3-way tensor using multiplicative update rule [17].

Algorithm 4: 3-way nonnegative Tucker-ALS

```
Input: Tensor $\mathcal{X} \in \mathbb{R}_+^{I\times J\times K}$, desired core size: $P \times Q \times R$
Output: Core tensor $\mathcal{G} \in \mathbb{R}_+^{P\times Q\times R}$ and orthogonal factor matrices
    $\mathbf{A} \in \mathbb{R}_+^{I\times P}, \mathbf{B} \in \mathbb{R}_+^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}_+^{K\times R}$
1: Initialize $\mathbf{B}, \mathbf{C}$;
2: repeat
3:     $\mathbf{A} \leftarrow \mathbf{A} * \frac{(\mathcal{X}\times_2\mathbf{B}^T\times_3\mathbf{C}^T)_{(1)}\mathbf{G}_{(1)}^T}{\mathbf{A}(\mathcal{G}\times_2\mathbf{B}^T\mathbf{B}\times_3\mathbf{C}^T\mathbf{C})_{(1)}\mathbf{G}_{(1)}^T}$;
4:     Normalize columns of $\mathbf{A}$;
5:     $\mathbf{B} \leftarrow \mathbf{B} * \frac{(\mathcal{X}\times_1\mathbf{A}^T\times_3\mathbf{C}^T)_{(2)}\mathbf{G}_{(2)}^T}{\mathbf{B}(\mathcal{G}\times_1\mathbf{A}^T\mathbf{A}\times_3\mathbf{C}^T\mathbf{C})_{(2)}\mathbf{G}_{(2)}^T}$;
6:     Normalize columns of $\mathbf{B}$;
7:     $\mathbf{C} \leftarrow \mathbf{C} * \frac{(\mathcal{X}\times_1\mathbf{A}^T\times_2\mathbf{B}^T)_{(3)}\mathbf{G}_{(3)}^T}{\mathbf{C}(\mathcal{G}\times_1\mathbf{A}^T\mathbf{A}\times_2\mathbf{B}^T\mathbf{B})_{(3)}\mathbf{G}_{(3)}^T}$;
8:     Normalize columns of $\mathbf{C}$;
9:     $\mathcal{G} \leftarrow \mathcal{G} * \frac{\mathcal{X}\times_1\mathbf{A}^T\times_2\mathbf{B}^T\times_3\mathbf{C}^T}{\mathcal{G}\times_1\mathbf{A}^T\mathbf{A}\times_2\mathbf{B}^T\mathbf{B}\times_3\mathbf{C}^T\mathbf{C}}$;
10: until $||\mathcal{G}||$ ceases to increase or the maximum number of outer
    iterations is exceeded.
```

3 Proposed Method

In this section, we present HATEN2, our proposed distributed MAPREDUCE algorithms for large scale tensor decompositions. Section 3.1 gives the main ideas of HATEN2; Section 3.2 describes details of HATEN2; and Section 3.3 extends HATEN2 for handling the nonnegativity-constraints.

3.1 Overview

How can we design scalable and efficient PARAFAC/Tucker decomposition algorithms for very large tensors? The most challenging parts of those algorithms are n-mode matrix product $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$ (Lines 3, 5, and 7 of Algorithms 2 and 4) in Tucker-ALS and nonnegative Tucker-ALS, and Khatri-Rao product $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$ (Lines 3, 5, and 7 of Algorithms 1 and 3) in PARAFAC-ALS and nonnegative PARAFAC-ALS. There are several challenges in designing efficient distributed algorithms for these operations.

– **Minimize intermediate data.** During the computation, huge intermediate data are generated in the shuffle stage. How can we minimize the intermediate data?
– **Minimize disk accesses.** How can we minimize the disk accesses to decrease the running time?
– **Minimize jobs.** How can we minimize the number of MAPREDUCE jobs to decrease the running time?

Our main ideas to address the challenges are as follows.

– **Decoupling the steps in n-mode vector product.** We decouple the multiplication and the addition steps in n-mode vector product by introducing a new operation called Hadamard-and-Merge which leads to decreasing the intermediate data size (Section 3.2.2).
– **Removing dependencies in sequential products.** We remove dependencies by carefully reordering the computations, and exploiting the sparsity of real world tensors. It leads to further decreasing the intermediate data size (Section 3.2.3).
– **Integrating jobs by increasing memory usage.** We integrate multiple MAPREDUCE jobs by increasing memory usage. The idea leads to minimizing the number of jobs and the disk accesses (Section 3.2.4).

Figure 4 shows the framework of our proposed HATEN2-DRI (or just HATEN2) method which contains all the above ideas. Note that although the computations for the two decompositions Tucker and PARAFAC are different, our HATEN2 unifies them into a general framework where the two methods differ only at the final merge step: HATEN2-Tucker uses $CrossMerge$, while HATEN2-PARAFAC uses $Pairwise\text{-}Merge$ (see Section 3.2.4 for details). In the next subsection, we describe the three main ideas in detail.

3.2 Method Details

In the following, we start with a naive method, and improve the method gradually by adding several ideas one by one until we reach the final method HATEN2-DRI (or just HATEN2). Figure 5 and Table 2 summarize the differences between all methods. Tables 3 and 4 show the total costs of all the methods in terms of the maximum intermediate data size, and the number of total MAPREDUCE jobs.

3.2.1 Naive Method

The most naive method is the straightforward implementation of the idea in MET [5], the state-of-the art single ma-

Table 2: Comparison of all methods experimented. Our proposed and recommended method is HATEN2-DRI (or just HATEN2) which incorporates all the proposed ideas.

Method	Distributed?	Decoupling the Steps (D/N) (Section 3.2.2)	Remove Dependencies (R/N) (Section 3.2.3)	Integrating Jobs (I/N) (Section 3.2.4)
Tensor Toolbox	No	No	No	No
HATEN2-Naive	Yes	No	No	No
HATEN2-DNN	Yes	Yes	No	No
HATEN2-DRN	Yes	Yes	Yes	No
HATEN2-DRI (or just HATEN2)	Yes	Yes	Yes	Yes

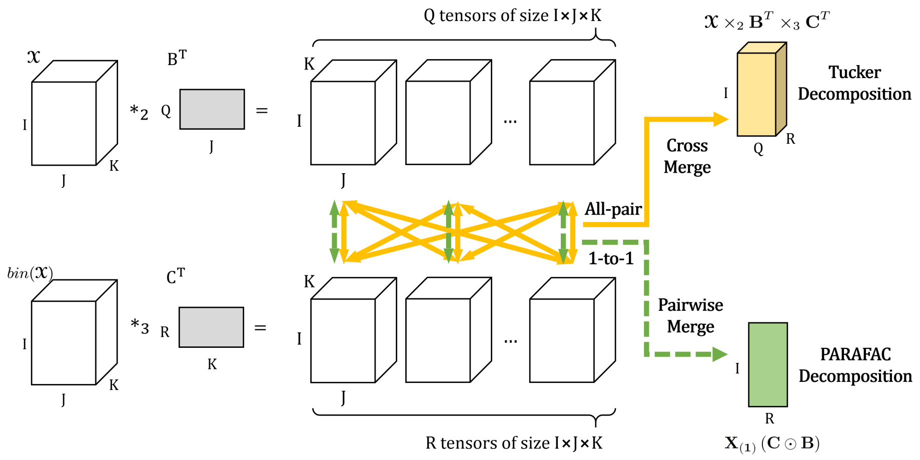


Fig. 4: General computational framework in HATEN2 for Tucker and PARAFAC decompositions ($Q = R$ in PARAFAC). Although the two decompositions are different, our HATEN2 unifies them into a general framework where the two methods differ only at the final merge step: $CrossMerge$ for HATEN2-Tucker, and $PairwiseMerge$ for HATEN2-PARAFAC (see Section 3.2.4 for details).

Table 3: Summary of costs in all methods for computing $\mathcal{X} \times_2 \mathbf{B} \times_3 \mathbf{C}$ in Tucker decomposition. We replace $nnz(\mathcal{X} \times_2 \mathbf{B})$ with $nnz(\mathcal{X})Q$ according to the estimation of $nnz(\mathcal{X} \times_2 \mathbf{B})$ in Lemma 4.

Method	Max. Intermediate Data	Total Jobs
HATEN2-Tucker-Naive	$nnz(\mathcal{X}) + IJK$	$Q + R$
HATEN2-Tucker-DNN	$nnz(\mathcal{X})QR$	$Q + R + 2$
HATEN2-Tucker-DRN	$nnz(\mathcal{X})(Q + R)$	$Q + R + 1$
HATEN2-Tucker-DRI	$nnz(\mathcal{X})(Q + R)$	2

chine implementation which was adopted by Tensor Toolbox. The main idea is to perform each n-mode vector product separately. HATEN2-Tucker-Naive computes $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ first by performing $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ operation Q times, and then computes $\mathcal{T} \times_3 \mathbf{C}^T$ by performing $\mathcal{T} \bar{\times}_3 \mathbf{c}_r^T$ operation R times, where $\mathbf{b}_q$ and $\mathbf{c}_r$ are the qth row of $\mathbf{B}$, and the rth row of $\mathbf{C}$, respectively. Algorithm 5 shows HATEN2-Tucker-Naive method.

Table 4: Summary of costs in all methods for computing $\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$ in PARAFAC decomposition.

Method	Max. Intermediate Data	Total Jobs
HATEN2-PARAFAC-Naive	$nnz(\mathcal{X}) + IJK$	$2R$
HATEN2-PARAFAC-DNN	$nnz(\mathcal{X}) + J$	$4R$
HATEN2-PARAFAC-DRN	$2nnz(\mathcal{X})R$	$2R + 1$
HATEN2-PARAFAC-DRI	$2nnz(\mathcal{X})R$	2

Similarly, HATEN2-PARAFAC-Naive computes $\mathcal{T}_r = \mathcal{X} \bar{\times}_2 \mathbf{b}_r^T$ first, and then computes $\mathcal{Y}_r = \mathcal{T}_r \bar{\times}_3 \mathbf{c}_r^T$. It computes $\mathcal{Y}$ by performing these operations R times. Algorithm 6 shows HATEN2-PARAFAC-Naive method.

MAPREDUCE algorithm. The MAPREDUCE algorithm of n-mode vector product $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ in HATEN2-Naive is as follows.

<**Naive:** $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ >

– **MAP**: map $< i, j, k, \mathcal{X}(i, j, k) >$ on $(iK + k)$, such that tuples with the same key are shuffled to the

Algorithm 5: HATEN2-Tucker-Naive for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times Q \times R}$
1: for q=1,..,Q do
2:     $\mathcal{T}_q \leftarrow \mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$;
3: end for
4: for r=1,..,R do
5:     $\mathcal{Y}_r \leftarrow \mathcal{T} \bar{\times}_3 \mathbf{c}_r^T$;
6: end for
```

Algorithm 6: HATEN2-PARAFAC-Naive for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times R}$
1: for r=1,..,R do
2:     $\mathcal{T}_r \leftarrow \mathcal{X} \bar{\times}_2 \mathbf{b}_r^T$;
3:     $\mathcal{Y}_r \leftarrow \mathcal{T}_r \bar{\times}_3 \mathbf{c}_r^T$;
4: end for
```

same reducer in the form of <key: $(iK + k)$, values: $\{(j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{i:k})\}$ >, and send $\mathbf{b}_q$ to the all the reducers in the form of <key: $(iK + k) | \forall (i,k)$, values: $\{(j, \mathbf{b}_q(j)) | \forall j \in idx(\mathbf{b}_q)\}$ >.

– **REDUCE**: take <key: $(iK + k)$, values: $\{(j, \mathbf{b}_q(j)) | \forall j \in idx(\mathbf{b}_q)\}$, $\{(j, \mathcal{X}(i,j,k)) | \forall (i,j,k) \in idx(\mathcal{X}_{i:k})\}$ >, and emit $< i, k, \sum_{j=1}^{J} \mathcal{X}(i,j,k)\mathbf{b}_q(j) >$.

The reducer processing the key $iK + k$ receives the non-zero elements of $\mathcal{X}_{i:k}$ and $\mathbf{b}_q$. Then it performs the inner product of the two vectors, and outputs an element of the result tensor $\mathcal{T}_q$. $\mathcal{T} \bar{\times}_3 \mathbf{c}_r^T$ operation is handled in the same manner. Although simple, this naive implementation has too much overhead because 1) the vector $\mathbf{b}_q^T$ is copied IK times which eventually generates too much intermediate data ($nnz(\mathcal{X}) + IJK$), and 2) the vector $\mathbf{b}_q^T$ might not fit in the memory of a machine when J is very large. How can we improve this naive method? In the following three subsections we incrementally improve the naive method.

3.2.2 Decoupling the Steps in n-mode Vector Product

The first idea to improve the naive method is to make the n-mode vector product $\bar{\times}_n$ scalable. As we saw in the previous subsection, the naive algorithm which broadcasts the vector $\mathbf{b}_q^T$ has too much overhead. Our idea, called Hadamard-and-Merge, is to decouple the product into two steps: the Hadamard product step where the element of the vector is multiplied with the corresponding element of the tensor, and the merge step where the multiplied values are summed. Hadamard-and-Merge operation comprises the two following operations: n-mode vector Hadamard product and *Collapse*.

Definition 1 **(n-mode vector Hadamard product)** The n-mode vector Hadamard product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X} \bar{*}_n \mathbf{v}$ and is of size $I_1 \times I_2 \times \cdots \times I_N$. It is defined by

$$(\mathcal{X} \bar{*}_n \mathbf{v})_{i_1 \ldots i_n \ldots i_N} = x_{i_1 \ldots i_n \ldots i_N} v_{i_n}.$$

Definition 2 **($Collapse(\mathcal{X})_n$)** The *Collapse* operation of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ on mode n is denoted by $Collapse(\mathcal{X})_n$ and is of size $I_1 \times \cdots \times I_{(n-1)} \times I_{(n+1)} \times \cdots \times I_N$. It is defined by

$$(Collapse(\mathcal{X})_n)_{i_1 \ldots i_{n-1} i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_n \ldots i_N}.$$

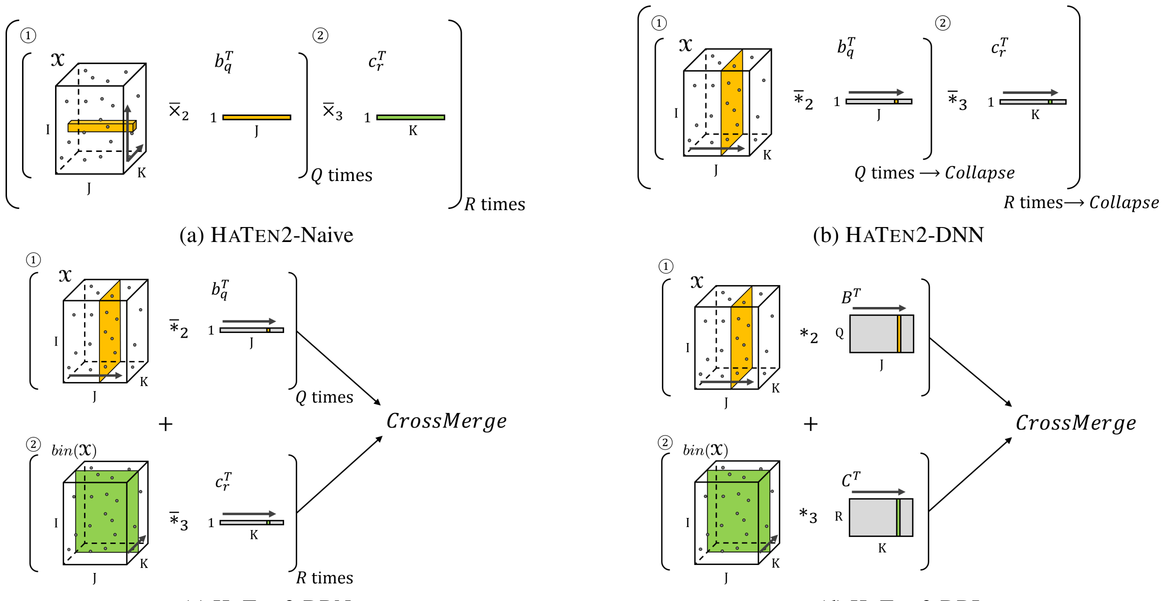
Intuitively, the n-mode vector Hadamard product is a generalization of Hadamard product of two vectors. The $Collapse(\mathcal{X})_n$ operation sums up all the values of a tensor $\mathcal{X}$ across the mode n. With these definitions, HATEN2-DNN expresses the original n-mode vector product $\mathcal{X} \bar{\times}_2 \mathbf{b}_q^T$ by $Collapse\ (\mathcal{X} \bar{*}_2 \mathbf{b}_q^T)_2$. By decoupling the n-mode vector product into two steps, HATEN2-DNN greatly decreases the intermediate data size of HATEN2-Naive from $nnz(\mathcal{X}) + IJK$ to $nnz(\mathcal{X})QR$ for Tucker, and from $nnz(\mathcal{X}) + IJK$ to $nnz(\mathcal{X}) + J$ for PARAFAC. Algorithms 7 and 8 show HATEN2-DNN for Tucker and PARAFAC decompositions, respectively. In HATEN2-Tucker-DNN, we compute $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T \in \mathbb{R}^{I \times Q \times K}$ by iteratively performing $\mathcal{T'}_q = \mathcal{X} \bar{*}_2 \mathbf{b}_q^T$ for Q times, and then merging them using the operation $Collapse(\mathcal{T'})_2$. $\mathcal{T} \times_3 \mathbf{C}^T$ operation is handled in the same manner. In HATEN2-PARAFAC-DNN, *Collapse* is applied right after the individual n-mode vector Hadamard product.

Algorithm 7: HATEN2-Tucker-DNN for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times Q \times R}$
1: for q=1,..,Q do
2:     $\mathcal{T'}_q \leftarrow \mathcal{X} \bar{*}_2 \mathbf{b}_q^T$;
3: end for
4: $\mathcal{T} \leftarrow Collapse(\mathcal{T'})_2$;
5: for r=1,..,R do
6:     $\mathcal{Y'}_r \leftarrow \mathcal{T} \bar{*}_3 \mathbf{c}_r^T$;
7: end for
8: $\mathcal{Y} \leftarrow Collapse(\mathcal{Y'})_3$;
```

MAPREDUCE algorithm. The MAPREDUCE algorithms of n-mode vector Hadamard product and $Collapse(\mathcal{X})_n$ in HATEN2-DNN are expressed as follows.

$< \mathcal{X} \bar{*}_2 \mathbf{b}_q^T >$



(a) HATEN2-Naive (b) HATEN2-DNN (c) HATEN2-DRN (d) HATEN2-DRI

Fig. 5: Comparison of all HATEN2 variants for Tucker decomposition. Areas with the same color are sent to the same reducer in the MAPREDUCE jobs for n-mode (Hadamard) product.

Algorithm 8: HATEN2-PARAFAC-DNN for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)}\,(\mathbf{C} \odot \mathbf{B})$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J \times R}$,
       $\mathbf{C} \in \mathbb{R}^{K \times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I \times R}$
1: for r=1,..,R do
2:     $\mathcal{T}'_r \leftarrow \mathcal{X} \bar{*}_2 \mathbf{b}_r^T$;
3:     $\mathcal{T}_r \leftarrow Collapse(\mathcal{T}'_r)_2$;
4:     $\mathcal{Y}'_r \leftarrow \mathcal{T}_r \bar{*}_3 \mathbf{c}_r^T$;
5:     $\mathcal{Y}_r \leftarrow Collapse(\mathcal{Y}'_r)_3$;
6: end for
```

- **MAP**: map $< \ i, j, k, \mathcal{X}(i,j,k) \ >$ on j, and $< j, q, \mathbf{b}_q(j) \ >$ on j such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $(q, \mathbf{b}_q(j)), \{(i,k,\mathcal{X}(i,j,k)) | \forall (i,k) \in idx(\mathcal{X}_{i:k})\} >$.
- **REDUCE**: take <key: j, values: $(q, \mathbf{b}_q(j)), \{(i,k,\mathcal{X}(i,j,k)) | \forall (i,k) \in idx(\mathcal{X}_{i:k})\} >$ and emit $< \ i, j, k, q, \mathcal{X}(i,j,k)\mathbf{b}_q(j) \ >$ for each $(i,k) \in idx(\mathcal{X}_{i:k})$.

$< Collapse(\mathcal{T})_2 >$

- **MAP**: map $< i, j, k, q, \mathcal{X}(i,j,k)\mathbf{B}(j,q) >$ on $(iK+k)$ such that tuples with the same key are shuffled to the same reducer in the form of <key: $(iK+k)$, values: $\{(q, \mathcal{X}(i,j,k)\mathbf{B}(j,q)) | \forall (i,j,k,q) \in idx(\mathcal{T}_{i:k:})\} >$.
- **REDUCE**: take <key: $(iK + k)$, values: $\{(q, \mathcal{X}(i,j,k)\mathbf{B}(j,q)) | \forall (i,j,k,q) \in idx(\mathcal{T}_{i:k:})\} >$ and emit $< \ i, q, k, \sum_j \mathcal{X}(i,j,k)\mathbf{B}(j,q) \ >$ for each $(i,j,k,q) \in idx(\mathcal{T}_{i:k:})$.

In n-mode vector Hadamard product, the mappers send $nnz(\mathcal{X}_{:j:})$ of j-th slice and an element of $\mathbf{b}_q$ to reducers using j as the key. The reducers multiply the vector element with the tensor elements. In *Collapse* operation, mappers send $nnz(\mathcal{X}_{i:k})Q$ elements to reducers using $iK + k$ as the key. The reducers aggregate the values.

3.2.3 Removing Dependencies in Sequential Products

The previous two methods HATEN2-Naive and HATEN2-DNN have a significant problem: they have dependencies in their computation sequences. In Tucker decomposition, to compute $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$, both of the previous methods first compute $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ by multiplying $\mathcal{X}$ and columns of B, and then multiply $\mathcal{T}$ with the columns of C. That is, the second step cannot be initiated until the first step is finished. Similarly, in PARAFAC decomposition, computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)}\,(\mathbf{C} \odot \mathbf{B})$ has a dependency: $\mathcal{X}$ is multiplied with $\mathbf{b}^T$ first, and the result is multiplied with $\mathbf{c}^T$. These dependencies in the computation have the following problems.

- Too large intermediate data: in Tucker decomposition, the number $nnz(\mathcal{T})$ of nonzero elements in $\mathcal{T} = \mathcal{X} \times_2 \mathbf{B}^T$ is estimated to be $nnz(\mathcal{X})Q$ for a sparse tensor $\mathcal{X}$, as described in Lemma 4. Thus, multiplying $\mathcal{T}$ with $\mathbf{C}^T$ would require intermediate data of size $nnz(\mathcal{X})QR$ which is prohibitively large.

– Too many MAPREDUCE jobs: in PARAFAC decomposition, HATEN2-PARAFAC-DNN requires $4R$ MAPREDUCE jobs, since the first multiplication with $\mathbf{b}^T$, and the second multiplication with $\mathbf{c}^T$ are performed in sequence.

Our idea to solve the problems is to remove the dependencies by carefully reordering the computations, and exploiting the sparsity of real world tensors. Before describing the details, we introduce two new operations $CrossMerge$ and $PairwiseMerge$ as follows.

Definition 3 ($CrossMerge$) The $CrossMerge$ operation of $N-1$ tensors $\mathcal{X}_1 \in \mathbb{R}^{I_1\times\ldots\times I_N\times J_1}$, ..., $\mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N \in \mathbb{R}^{I_1\times\ldots\times I_N\times J_N}$ on the mode n is denoted by $CrossMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)}$ and is of size $\mathbb{R}^{I_n\times J_1\times\ldots\times J_N}$. It is defined by

$$(CrossMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)})_{i_n j_1 \ldots j_N} = \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)=(1,\ldots,1)}^{I_1,\ldots,I_{n-1},I_{n+1},\ldots,I_N} \mathcal{X}_1(i_1, ..., i_N, j_1)\times\cdots\times\mathcal{X}_m(i_1, ...i_N, j_N),$$

for all $j_i = 1, ..., J_i$ where $i \neq n$.

Definition 4 ($PairwiseMerge$) The $PairwiseMerge$ operation of $N-1$ tensors $\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N \in \mathbb{R}^{I_1\times\ldots\times I_N\times J}$ on the mode n is denoted by $PairwiseMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)}$ and is of size $\mathbb{R}^{I_n\times J}$. It is defined by

$$(PairwiseMerge(\mathcal{X}_1, ..., \mathcal{X}_{n-1}, \mathcal{X}_{n+1}, ..., \mathcal{X}_N)_{(n)})_{i_n j} = \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)=(1,\ldots,1)}^{I_1,\ldots,I_{n-1},I_{n+1},\ldots,I_N} \mathcal{X}_1(i_1, ..., i_N, j)\times\cdots\times\mathcal{X}_m(i_1, ...i_N, j),$$

for all $j = 1, ..., J$.

Our crucial observation is that these two operations can be used for removing the dependencies in the computations of $\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$ and $\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$, respectively, as shown in the following lemmas.

Lemma 2 (CrossMerge) *Given $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, $\mathbf{B} \in \mathbb{R}^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$,*

$$\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T \Leftrightarrow CrossMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$$

where $\mathcal{T}' \in \mathbb{R}^{I\times J\times K\times Q}$ is a tensor whose qth subtensor $\mathcal{T}'_{:::q}$ is given by $\mathcal{X}\bar{*}_2\mathbf{b}_q^T$, and $\mathcal{T}'' \in \mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\mathcal{T}'_{:::r}$ is given by $bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$.

Proof. The (i, q, k)-th element $\mathcal{M}_{iqk}$ of $\mathcal{M} = \mathcal{X} \times_2 \mathbf{B}^T$ is given by

$$\mathcal{M}_{iqk} = \sum_{j=1}^{J} \mathcal{X}(i, j, k)\mathbf{B}(j, q).$$

Then the (i, q, r)-th element of $(\mathcal{X} \times_2 \mathbf{B}^T) \times_3 \mathbf{C}^T$ is

$$\sum_{k=1}^{K} \mathcal{M}(i, q, k)\mathbf{C}(k, r)$$

$$= \sum_{k=1}^{K}(\sum_{j=1}^{J} \mathcal{X}(i, j, k)\mathbf{B}(j, q))\mathbf{C}(k, r)$$

$$= \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i, j, k)\mathbf{B}(j, q)\mathbf{C}(k, r) \quad (1)$$

The (i, j, k, q)-th element of subtensor $\mathcal{T}'_{:::q}$ is given by $\mathcal{X}(i, j, k)\mathbf{b}_q^T(j)$, and (i, j, k, r)-th element of subtensor $\mathcal{T}''_{:::r}$ is given by $(bin(\mathcal{X})(i, j, k))\mathbf{c}_r^T(k)$. Therefore, the (i, j, k, q)-th element of $\mathcal{T}'$ is

$$\mathcal{T}'_{ijkq} = \mathcal{X}(i, j, k)\mathbf{B}(j, q),$$

and the (i, j, k, r)-th element of $\mathcal{T}''$ is

$$\mathcal{T}''_{ijkr} = (bin(\mathcal{X})(i, j, k))\mathbf{C}(k, r).$$

The (i, q, r)-th element of $CrossMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$ is

$$\sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{T}'(i, j, k, q)\mathcal{T}''(i, j, k, r).$$

$$= \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i, j, k)\mathbf{B}(j, q)(bin(\mathcal{X})(i, j, k))\mathbf{C}(k, r)$$

$$= \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i, j, k)\mathbf{B}(j, q)\mathbf{C}(k, r) \quad (2)$$

where the last equality uses the fact $\mathcal{X}(i, j, k) \times (bin(\mathcal{X})(i, j, k)) = \mathcal{X}(i, j, k)$. The equation (1) for (i, q, r)-th element of $\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$ is exactly the same as the equation (2) for (i, q, r)-th element of $CrossMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$. ■

Lemma 3 (PairwiseMerge) *Given $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$,*

$$\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}) \Leftrightarrow PairwiseMerge(\mathcal{F}', \mathcal{T}'')_{(1)}$$

where $\mathcal{F}' \in \mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\mathcal{F}'_{:::r}$ is given by $\mathcal{X}\bar{*}_2\mathbf{b}_r^T$, and $\mathcal{T}'' \in \mathbb{R}^{I\times J\times K\times R}$ is a tensor whose rth subtensor $\mathcal{T}'_{:::r}$ is given by $bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$.

Proof. The (i, r)-th element of $\mathbf{M} = \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$ is defined by

$$\mathbf{M}_{ir} = \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i, j, k)\mathbf{B}(j, r)\mathbf{C}(k, r) \quad (3)$$

The (i, j, k)-th element of subtensor $\mathcal{F}'_{:::r}$ is given by $\mathcal{X}(i, j, k)\mathbf{b}_r^T(j)$, and the (i, j, k)-th element of subtensor

$\mathcal{T}'_{:::r}$ is given by $(bin(\mathcal{X})(i,j,k))\mathbf{c}_r^T(k)$. Therefore, the (i,j,k,r)-th element of $\mathcal{F}'$ is

$$\mathcal{F}'_{ijkr} = \mathcal{X}(i,j,k)\mathbf{B}(j,r),$$

and the (i,j,k,r)-th element of $\mathcal{T}''$ is

$$\mathcal{T}''_{ijkr} = (bin(\mathcal{X})(i,j,k))\mathbf{C}(k,r).$$

The (i,r)-th element of $PairwiseMerge(\mathcal{F}',\mathcal{T}'')_{(1)}$ is

$$\sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{F}'(i,j,k,r)\mathcal{T}''(i,j,k,r).$$

$$= \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i,j,k)\mathbf{B}(j,r)(bin(\mathcal{X})(i,j,k))\mathbf{C}(k,r)$$

$$= \sum_{(j,k)=(1,1)}^{(J,K)} \mathcal{X}(i,j,k)\mathbf{B}(j,r)\mathbf{C}(k,r)\cdots(2). \tag{4}$$

where the last equality uses the fact $\mathcal{X}(i,j,k) \times (bin(\mathcal{X})(i,j,k)) = \mathcal{X}(i,j,k)$. The equation (3) for (i,r)-th element of $\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$ is exactly the same as the equation (4) for (i,r)-th element of $PairwiseMerge(\mathcal{F}',\mathcal{T}'')_{(1)}$. ■

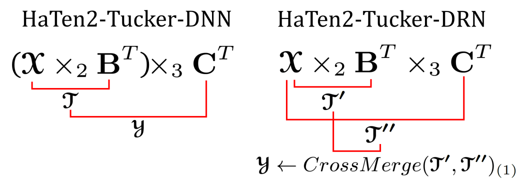

HaTen2-Tucker-DNN
HaTen2-Tucker-DRN
$(\mathcal{X} \times_2 \mathbf{B}^T)\times_3 \mathbf{C}^T$
$\mathcal{T}$
$\mathcal{Y}$
$\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$
$\mathcal{T}'$
$\mathcal{T}''$
$\mathcal{Y} \leftarrow CrossMerge(\mathcal{T}',\mathcal{T}'')_{(1)}$

Fig. 6: Comparison of HATEN2-Tucker-DNN and HATEN2-Tucker-DRN.

Using these two operations, HATEN2-DRN-Tucker, shown in Algorithm 9, computes $\mathcal{T}'$ and $\mathcal{T}''$ first, and then merges the result. Figure 6 illustrates the difference of HATEN2-Tucker-DRN and HATEN2-Tucker-DNN. Note that both $\mathcal{T}'$ and $\mathcal{T}''$ are sparse if the input tensor $\mathcal{X}$ is sparse, which is true in most real world tensors. Thus, HATEN2-Tucker-DRN further decreases the intermediate data size of HATEN2-DNN from $nnz(\mathcal{X})QR$ to $nnz(\mathcal{X})(Q+R)$. We want to emphasize that this decrease of the intermediate data size comes from the sparsity of real world tensors where $nnz(\mathcal{X}) \sim I$; if the input tensor is a full tensor (which is not realistic), the intermediate data size of HATEN2-DNN becomes $nnz(\mathcal{X})Q$ which is smaller than that of HATEN2-DRN. Also, note that the removal of dependency in HATEN2-DRN enables computing $\mathcal{T}'$ and $\mathcal{T}''$ in parallel; the idea is reflected in HATEN2-DRI (see Section 3.2.4 for details). For PARAFAC decomposition, HATEN2-PARAFAC-DRN, shown in Algorithm 10, decreases the number of MAPREDUCE jobs of HATEN2-PARAFAC-DNN from $4R$ to $2R+1$.

Algorithm 9: HATEN2-Tucker-DRN for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J\times Q}$,
    $\mathbf{C} \in \mathbb{R}^{K\times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I\times Q\times R}$
1: for q=1,...,Q do
2:     $\mathcal{T}'_q \leftarrow \mathcal{X}\bar{*}_2\mathbf{b}_q^T$;
3: end for
4: for r=1,...,R do
5:     $\mathcal{T}''_r \leftarrow bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$;
6: end for
7: $\mathcal{Y} \leftarrow CrossMerge(\mathcal{T}',\mathcal{T}'')_{(1)}$;
```

Algorithm 10: HATEN2-PARAFAC-DRN for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J\times R}$,
    $\mathbf{C} \in \mathbb{R}^{K\times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I\times R}$
1: for r=1,...,R do
2:     $\mathcal{F}'_r \leftarrow \mathcal{X}\bar{*}_2\mathbf{b}_r^T$;
3: end for
4: for r=1,...,R do
5:     $\mathcal{T}''_r \leftarrow bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$;
6: end for
7: $\mathcal{Y} \leftarrow PairwiseMerge(\mathcal{F}',\mathcal{T}'')_{(1)}$;
```

MAPREDUCE algorithm. The MAPREDUCE algorithms of $CrossMerge(\mathcal{T}',\mathcal{T}'')_{(1)}$ and $PairwiseMerge(\mathcal{F}',\mathcal{T}'')_{(1)}$ operations are as follows.

$< CrossMerge(\mathcal{T}',\mathcal{T}'')_{(1)} >$

- **MAP**: map $< i,j,k,q,\mathcal{X}(i,j,k)\mathbf{b}_q(j) >$ on $(i, rQ+q)$ for all $r = 1,\ldots R$, and $< i,j,k,r,\mathbf{c}_r(k) >$ on $(i, rQ+q)$ for all $q = 1,\ldots Q$ such that tuples with the same key are shuffled to the same reducer in the form of <key: $(i, rQ+q)$, values: $\{(j,k,\mathcal{X}(i,j,k)\mathbf{b}_q(j))|\forall(i,j,k) \in idx(\mathcal{X}_{i::})\}, \{(j,k,r,\mathbf{c}_r(k))|\forall(i,j,k) \in idx(\mathcal{X}_{i::})\} >$.
- **REDUCE**: take <key: $(i, rQ+q)$, values: $\{(j,k,\mathcal{X}(i,j,k)\mathbf{b}_q(j))|\forall(i,j,k) \in idx(\mathcal{X}_{i::})\}$, $\{(j,k,r,\mathbf{c}_r(k))|\forall(i,j,k) \in idx(\mathcal{X}_{i::})\} >$ and emit $< \{i,q,r,\sum_{j,k}\mathcal{X}(i,j,k)\mathbf{b}_q(j)\mathbf{c}_r(k)$ for all q=1, ..., Q, r=1, ..., $R\} >$ for each $(i,j,k) \in idx(\mathcal{X}_{i::})$.

$< PairwiseMerge(\mathcal{F}',\mathcal{T}'')_{(1)} >$

- **MAP**: map $< i,j,k,r,\mathcal{X}(i,j,k)\mathbf{b}_r(j) >$ on (i,r), and $< i,j,k,r,\mathbf{c}_r(k) >$ on (i,r) such that tuples with the same key are shuffled to the same reducer in the form of <key: (i, r), values: $\{(j, k, \mathcal{X}(i, j, k)\mathbf{b}_r(j)) |\forall(i, j, k) \in idx(\mathcal{X}_{i::})\}, \{(j, k, \mathbf{c}_r(k))|\forall(i, j, k) \in idx(\mathcal{X}_{i::})\}$ >.

- **REDUCE**: take <key: (i, r), values: $\{(j, k, \mathcal{X}(i, j, k)\mathbf{b}_r(j))|\forall(i, j, k) \in idx(\mathcal{X}_{i::})\}$, $\{(j, k, \mathbf{c}_r(k))|\forall(i, j, k) \in idx(\mathcal{X}_{i::})\}$ > and emit < $\{i, r, \sum_{j,k} \mathcal{X}(i, j, k)\mathbf{b}_r(j)\mathbf{c}_r(k)$ for all $r = 1, ..., R\}$ > for each $(i, j, k) \in idx(\mathcal{X}_{i::})$.

3.2.4 Integrating Jobs by Increasing Memory Usage

Although HATEN2-DRN decreased the intermediate data size and the number of MAPREDUCE jobs, the number of jobs is still significant: it is $Q + R + 1$ for HATEN2-Tucker-DRN and $2R + 1$ for HATEN2-PARAFAC-DRN. In this subsection, we propose HATEN2-DRI to further decrease the number of jobs to 2, thereby decreasing the disk accesses and the running time. Algorithms 11 and 12 show HATEN2-Tucker-DRI, and HATEN2-PARAFAC-DRI, respectively. HATEN2-DRI has two main ideas: integrating 1) vector products into a matrix product, and 2) products for different factor matrices.

Algorithm 11: HATEN2-Tucker-DRI for computing $\mathcal{Y} \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J\times Q}$,
    $\mathbf{C} \in \mathbb{R}^{K\times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I\times Q\times R}$
1: $(\mathcal{T}', \mathcal{T}'') \leftarrow IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$;
2: $\mathcal{Y} \leftarrow CrossMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$;
```

Algorithm 12: HATEN2-PARAFAC-DRI for computing $\mathcal{Y} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and factor matrices $\mathbf{B} \in \mathbb{R}^{J\times R}$,
    $\mathbf{C} \in \mathbb{R}^{K\times R}$
Output: Tensor $\mathcal{Y} \in \mathbb{R}^{I\times R}$
1: $(\mathcal{T}', \mathcal{T}'') \leftarrow IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$;
2: $\mathcal{Y} \leftarrow PairwiseMerge(\mathcal{T}', \mathcal{T}'')_{(1)}$;
```

Integrating vector products into a matrix product. HATEN2-DRI performs several n-mode vector Hadamard products together in one MAPREDUCE job, instead of multiple jobs, using the n-mode matrix Hadamard product which we define as follows.

Definition 5 **(n-mode matrix Hadamard product)** The n-mode matrix Hadamard product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{Q\times I_n}$ is denoted by $\mathcal{X} *_n \mathbf{U}$ and is of size $I_1 \times I_2 \times \cdots \times I_N \times Q$. It is defined by

$$(\mathcal{X} *_n \mathbf{U})_{i_1 i_2 \dots i_N q} = (\mathcal{X} \bar{*}_n \mathbf{U}^T_{q:})_{i_1 i_2 \dots i_N}.$$

where $\mathbf{U}_{q:}$ is the qth row of $\mathbf{U}$.

MAPREDUCE algorithm. The MAPREDUCE algorithm of n-mode matrix Hadamard product is as follows:
$< \mathcal{X} *_2 \mathbf{B} >$

- **MAP**: map < $i, j, k, \mathcal{X}(i, j, k)$ > on j, and < $j, q, \mathbf{B}(j, q)$ > on j such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $\{(q, \mathbf{B}(j, q))|\forall q \in \{1, ..., Q\}\}$, $\{(i, k, \mathcal{X}(i, j, k))|\forall(i, j, k) \in idx(\mathcal{X}_{:j:})\}$ >.
- **REDUCE**: take <key: j, values: $\{(q, \mathbf{B}(j, q))|\forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i, j, k))|\forall(i, j, k) \in idx(\mathcal{X}_{:j:})\}$ > and emit < $i, j, k, q, \mathcal{X}(i, j, k)\mathbf{B}(j, q)$ > for each $(i, j, k) \in idx(\mathcal{X}_{:j:})$ and $q \in \{1, ..., Q\}$.

The previous HATEN2-DRN method performs the n-mode vector Hadamard product $\mathcal{X} \bar{*}_2 \mathbf{b}_q^T$ for Q times using $nnz(\mathcal{X}_{:j:}) + 1$ of memory space per reducer; however, our new method HATEN2-DRI performs $\mathcal{X} *_2 \mathbf{B}^T$ only once using $nnz(\mathcal{X}_{:j:}) + Q$ of memory space per reducer where Q is used for storing a column of B^T. HATEN2-DRI decreases the number of jobs significantly without introducing too much overhead since Q is very small (e.g., 10 or 20).

Integrating products for different factor matrices. HATEN2-DRI also integrates $\mathcal{X} *_2 \mathbf{B}^T$ and $bin(\mathcal{X}) *_3 \mathbf{C}^T$ computations into one MAPREDUCE job. This integration is possible since the dependency of the two operations is removed in HATEN2-DRN (and, hence in HATEN2-DRI). Thanks to the integration, the original tensor data $\mathcal{X}$ needs to be read from disks only once (not twice as in previous methods), and thus we further decrease the running time.

MAPREDUCE algorithm. The MAPREDUCE algorithm of the integrating operation, denoted by $IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$, is as follows. $IMHP(\mathcal{X}, \mathbf{B}, \mathbf{C})$

- **MAP**: map $< i, j, k, \mathcal{X}(i, j, k) >$ on j, $< j, q, \mathbf{B}(j, q) >$ on j, $< i, j, k, \mathcal{X}(i, j, k) >$ on k, and $< k, r, \mathbf{C}(k, r) >$ on k such that tuples with the same key are shuffled to the same reducer in the form of <key: j, values: $\{(q, \mathbf{B}(j, q))|\forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i, j, k)) |\forall(i, j, k) \in idx(\mathcal{X}_{:j:})\}$ >, and <key: k, values: $\{(r, \mathbf{C}(k, r))|\forall r \in \{1, ..., R\}\}, \{(i, j, \mathcal{X}(i, j, k)) |\forall(i, j, k) \in idx(\mathcal{X}_{:k:})\}$ >.
- **REDUCE**: take <key: j, values: $\{(q, \mathbf{B}(j, q))|\forall q \in \{1, ..., Q\}\}, \{(i, k, \mathcal{X}(i, j, k))| \forall(i, j, k) \in idx(\mathcal{X}_{:j:})\}$ > and emit < $i, j, k, q, \mathcal{X}(i, j, k)\mathbf{B}(j, q)$ > for each $(i, j, k) \in idx(\mathcal{X}_{:j:})$ and $q \in \{1, ..., Q\}$, and take <key: k, values: $\{(r, \mathbf{C}(k, r))|\forall r \in \{1, ..., R\}\}$,

$\{(i, j, \mathcal{X}(i, j, k))|\forall(i, j, k) \in idx(\mathcal{X}_{:k:})\} >$ and emit $< i, j, k, r, \mathbf{C}(k, r) >$ for each $(i, j, k) \in idx(\mathcal{X}_{:k:})$ and $r \in \{1, ..., R\}$.

Finally, we note that although the two decompositions Tucker and PARAFAC are different, our HATEN2-DRI unifies them in a general framework of IMHP and merge. As seen in Algorithms 11 and 12, as well as in Figure 4, HATEN2-Tucker-DRI and HATEN2-PARAFAC-DRI differ only in the merge function ($CrossMerge$ for HATEN2-Tucker-DRI, and $PairwiseMerge$ for HATEN2-PARAFAC-DRI). This general framework allows easier extension of the method for other algorithms, as well as simple maintenance of the code.

3.2.5 Cost of HATEN2

We present the costs of the steps of all HATEN2 methods in terms of the intermediate data size, the number of MAPREDUCE jobs, and the number of the floating point operations in Tables 5 and 6. Based on Tables 5 and 6, we compare the total costs of all the methods in terms of the maximum intermediate data size, and the number of total MAPREDUCE jobs in Tables 3 and 4. We replace $nnz(\mathcal{T} = \mathcal{X} \times_2 \mathbf{B})$ with $nnz(\mathcal{X})Q$ according to the estimation of $nnz(\mathcal{X} \times_2 \mathbf{B})$ in Lemma 4.

Lemma 4 *Given a sparse $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, and a fully dense $\mathbf{B} \in \mathbb{R}^{J\times Q}$, the first-order Taylor approximation of the number of nonzeros in $\mathcal{X} \times_2 \mathbf{B}$ is $nnz(\mathcal{X})Q$.*

Proof. Let $P(\mathcal{X}_{ijk})$ be the probability that $\mathcal{X}_{ijk} \neq 0$. Assuming uniform distribution, $P(\mathcal{X}_{ijk})$ is estimated to be $\frac{nnz(\mathcal{X})}{IJK}$. Since $\mathbf{B}$ is a fully-dense matrix, a nonzero element in $\mathcal{X}_{i:k}$ fiber, when multiplied with $\mathbf{B}$, appears as Q nonzero elements in the result tensor $\mathcal{X} \times_2 \mathbf{B}$. Thus, the probability that there is no element in the (i, k)-th fiber of $\mathcal{X} \times_2 \mathbf{B}$ is given by $Q(1 - P(\mathcal{X}_{ijk}))^J = Q(1 - \frac{nnz(\mathcal{X})}{IJK})^J$. Then the estimated number of nonzero elements in $\mathcal{X} \times_2 \mathbf{B}$ is given by $(1 - (1 - \frac{nnz(\mathcal{X})}{IJK})^J) \times IQK$. Applying the first-order Taylor expansion of $(1 + x)^n \approx 1 + nx$ to the equation $(1 - \frac{nnz(\mathcal{X})}{IJK})^J$, we get

$$(1 - (1 - \frac{nnz(\mathcal{X})}{IJK})^J) \times IQK \approx$$
$$(1 - (1 - J\frac{nnz(\mathcal{X})}{IJK})) \times IQK = nnz(\mathcal{X})Q$$

■

Note that in Tucker decomposition, HATEN2-Tucker-DRI which contains all the proposed ideas, has the minimum intermediate data size. In PARAFAC, the intermediate data size of HATEN2-PARAFAC-DNN seems smaller than that of HATEN2-PARAFAC-DRI. However, HATEN2-PARAFAC-DNN has a lower scalability since the matrix $\mathcal{T}_r$ becomes dense, and thus $\mathcal{T}_r \bar{*}_3 \mathbf{c}_r^T$ might raise an out of memory error. In contrast, HATEN2-PARAFAC-DRI scales well by exploiting the sparsity of real-world tensors with the idea described in Sections 3.2.3 and 3.2.4. For both decompositions, HATEN2-DRI has the minimum number of jobs.

3.3 Extensions of HATEN2 for Nonnegativity Constraints

In this section, we extend HATEN2 for nonnegativity constraints. We first propose HATEN2-PARAFACNN, a distributed nonnegative PARAFAC decomposition whose single machine version is shown in Algorithm 3. Then, we propose HATEN2-TuckerNN, a distributed nonnegative Tucker decomposition whose single machine version is shown in Algorithm 4. Similar to the standard PARAFAC and Tucker, the challenges in designing distributed nonnegative factorization algorithms are computing $\mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B})$ and $\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$, respectively, which incur the intermediate data explosion problem. Using the same ideas presented in Section 3, we reduce the size of the intermediate data as well as the number of the total jobs.

3.3.1 HATEN2-*PARAFACNN*

MapReduce Algorithm for HATEN2-PARAFACNN. Updating each factor of A, B, and C in Lines 3, 5, and 7 of Algorithm 3 requires four computational steps:
<Updating factor $\mathbf{A}$ >

- Step 1: $\mathbf{M}_1 \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$.
- Step 2: $\mathbf{M}_2 \leftarrow \mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}$.
- Step 3: $\mathbf{M}_3 \leftarrow \frac{1}{\mathbf{AM}_2}$.
- Step 4: $\mathbf{A} \leftarrow \mathbf{A} * \mathbf{M}_1 * \mathbf{M}_3$.

Step 1 uses the same MapReduce algorithm described in Section 3.2. In Step 2, the operations C^TC and B^TB utilize the parallel outer product technique of GigaTensor [18]. Since results of both operations are $R \times R$ matrices and R is small, Hadamard product becomes straightforward. Step 3 computes $\mathbf{A} * \mathbf{M_2}$ where $\mathbf{M}_2$ is $\mathbf{C}^T\mathbf{C}*\mathbf{B}^T\mathbf{B}$. Using the distributed cache multiplication described in GigaTensor [18], each machine receives $\mathbf{M}_2$ and calculates each row of the result matrix $\frac{1}{\mathbf{AM}_2}$ with the following map-only job.

- MAP-3: map $< i, \mathbf{A}(i, :) >$ on i and calculate the i-th row of $\frac{1}{\mathbf{AM}_2}$ with distributively cached $\mathbf{M}_2$ so that the i-th row of $\frac{1}{\mathbf{AM}_2}$ is produced by each mapper.

Step 4 computes the Hadamard product of the three matrices $\mathbf{A}$, $\mathbf{M}_1$, and $\mathbf{M}_3$.

- MAP-4: map $< i, \mathbf{A}(i, :) >$, $< i, \mathbf{M}_1(i, :) >$, and $< i, \mathbf{M}_3(i, :) >$ on i such that tuples with the same i are shuffled to the same reducer in the form of $< i, \{(\mathbf{A}(i, j), \mathbf{M}_1(i, j), \mathbf{M}_3(i, j)|\forall j)\} >$.
- REDUCE-4: take $< i, \{(\mathbf{A}(i, j), \mathbf{M}_1(i, j), \mathbf{M}_3(i, j)|\forall j)\} >$, and emit $< i, \mathbf{A}(i, :) * \mathbf{M}_1(i, :) * \mathbf{M}_3(i, :) >$.

Table 5: Summary of costs in the steps of all methods for computing $\mathcal{X} \times_2 \mathbf{B} \times_3 \mathbf{C}$ in Tucker decomposition. $\mathcal{T}$ denotes $\mathcal{X} \times_2 \mathbf{B}$.

Method	Step	Intermediate Data	Jobs
HATEN2-Tucker-Naive	$\mathcal{X}\bar{\times}_2\mathbf{b}_q^T$	$nnz(\mathcal{X}) + IJK$	Q
	$\mathcal{T}\bar{\times}_3\mathbf{c}_r^T$	$nnz(\mathcal{T}) + IQK$	R
HATEN2-Tucker-DNN	$\mathcal{X}\bar{*}_2\mathbf{b}_q^T$	$nnz(\mathcal{X}) + J$	Q
	$Collapse$	$nnz(\mathcal{X})Q$	1
	$\mathcal{T}\bar{*}_3\mathbf{c}_r^T$	$nnz(\mathcal{X})Q + K$	R
	$Collapse$	$nnz(\mathcal{X})QR$	1
HATEN2-Tucker-DRN	$\mathcal{X}\bar{*}_2\mathbf{b}_q^T$	$nnz(\mathcal{X}) + J$	Q
	$bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$	$nnz(\mathcal{X}) + K$	R
	$CrossMerge$	$nnz(\mathcal{X})Q + nnz(\mathcal{X})R$	1
HATEN2-Tucker-DRI	$IMHP$	$2nnz(\mathcal{X}) + JQ + KR$	1
	$CrossMerge$	$nnz(\mathcal{X})Q + nnz(\mathcal{X})R$	1

Table 6: Summary of costs in the steps of all methods for computing $\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$ in PARAFAC decomposition. $\mathcal{T}_r$ denotes $\mathcal{X}\bar{\times}_2\mathbf{b}_r^T$.

Method	Step	Intermediate Data	Jobs
HATEN2-PARAFAC-Naive	$\mathcal{X}\bar{\times}_2\mathbf{b}_r^T$	$nnz(\mathcal{X}) + IJK$	R
	$\mathcal{T}_r\bar{\times}_3\mathbf{c}_r^T$	$nnz(\mathcal{T}_r) + IK$	R
HATEN2-PARAFAC-DNN	$\mathcal{X}\bar{*}_2\mathbf{b}_r^T$	$nnz(\mathcal{X}) + J$	R
	$Collapse$	$nnz(\mathcal{X})$	R
	$\mathcal{T}_r\bar{*}_3\mathbf{c}_r^T$	$nnz(\mathcal{T}_r) + K$	R
	$Collapse$	$nnz(\mathcal{T}_r)$	R
HATEN2-PARAFAC-DRN	$\mathcal{X}\bar{*}_2\mathbf{b}_r^T$	$nnz(\mathcal{X}) + J$	R
	$bin(\mathcal{X})\bar{*}_3\mathbf{c}_r^T$	$nnz(\mathcal{X}) + K$	R
	$PairwiseMerge$	$2nnz(\mathcal{X})R$	1
HATEN2-PARAFAC-DRI	$IMHP$	$2nnz(\mathcal{X}) + JR + KR$	1
	$PairwiseMerge$	$2nnz(\mathcal{X})R$	1

3.3.2 HATEN2-*TuckerNN*

MapReduce Algorithm for HATEN2-TuckerNN. Updating each factor of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ in Lines 3, 5, and 7 of Algorithm 4 requires five computational steps:
<Updating factor $\mathbf{A}$ >

– Step 1: $\mathbf{M}_1 \leftarrow \mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T$.
– Step 2: $\mathbf{M}_2 \leftarrow (\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T)_{(1)}\mathbf{G}_{(1)}^T$.
– Step 3: $\mathbf{M}_3 \leftarrow (\mathcal{G} \times_2 \mathbf{B}^T\mathbf{B} \times_3 \mathbf{C}^T\mathbf{C})_{(1)}\mathbf{G}_{(1)}^T$.
– Step 4: $\mathbf{M}_4 \leftarrow \frac{1}{\mathbf{A}\mathbf{M}_3}$.
– Step 5: $\mathbf{A} \leftarrow \mathbf{A} * \mathbf{M}_2 * \mathbf{M}_4$.

Step 1 uses the same MapReduce algorithm as described in Section 3.2. Step 2 multiplies $(\mathcal{X} \times_2 \mathbf{B}^T \times_3 \mathbf{C}^T)_{(1)}$ and $\mathbf{G}_{(1)}^T$ which requires a similar MapReduce job of Step 3 in HATEN2-PARAFACNN. In Step 3, the operations $\mathbf{B}^T\mathbf{B}$ and $\mathbf{C}^T\mathbf{C}$ utilize the parallel outer product technique of GigaTensor [18] as explained in Step 2 of HATEN2-PARAFACNN. Since the sizes of the core tensor, $\mathbf{B}^T\mathbf{B}$, and $\mathbf{C}^T\mathbf{C}$ are small, we perform $(\mathcal{G} \times_2 \mathbf{B}^T\mathbf{B} \times_3 \mathbf{C}^T\mathbf{C})_{(1)}\mathbf{G}_{(1)}^T$ in a local machine. Steps 4 and 5 are the same as those in Steps 3 and 4 of HATEN2-PARAFACNN, respectively. Updating the core tensor $\mathcal{G}$ in Line 9 of Algorithm 4 requires similar computation steps.

4 Experiment

In this section, we present experimental results to answer the following questions.

Q1 What is the performance of HATEN2-DRI compared with other methods?
Q2 How well does HATEN2-DRI scale up with various factors (nonzeros, dimensionality, density, core tensor size, order, and machines)?

After describing the experimental settings in Section 4.1, we present the scalability results in Section 4.2 to answer **Q1** and **Q2**.

Table 7: Summary of the tensor data used. B: billion, M: million, K: thousand.

Data	I	J	K	Nonzeros	Density	Format	Source
Freebase-sampled	38 M	38 M	532	139 M	1.733×10^{-10}	{entity, entity, relation}	Freebase [19]
Phonecall	30M	30M	62	184M	3.297×10^{-9}	{sender, receiver, time}	
NELL	26 M	26 M	48 M	144 M	4.387×10^{-15}	{subject, object, predicate}	'Read the Web' [1]
NELL-2	15 K	15 K	29 K	77 M	1.262×10^{-5}	{subject, object, predicate}	'Read the Web' [1]
DARPA1998	22K	22K	23M	28M	2.515×10^{-9}	{source IP, destination IP, time}	MIT Lincoln Lab. [20]
Random	1 K∼100 M	1 K∼100 M	1 K∼100 M	10 K∼10 B	$10^{-15} \sim 10^{-5}$		

4.1 Experimental Settings

We compare our final method HATEN2-DRI with other methods (HATEN2-Naive, HATEN2-DNN, HATEN2-DRN) as well as the Tensor Toolbox [21], the state of the art tensor computation package for a single machine.

4.1.1 Machines

HATEN2 is run on a HADOOP cluster with 40 machines where each machine has a quad-core Intel Xeon E3 1230v3 3.3Ghz CPU, 32 GB RAM, and 12 Terabytes disk. The Tensor Toolbox is run on a machine from the HADOOP cluster.

4.1.2 Dataset

The tensor dataset used in our experiments is summarized in Table 7, with the following details.

– Freebase-sampled: sampled RDF triples (subject entity, object entity, relation) where the entries are related with music, book, tv shows, film, people, and sport from Freebase [19].
– Phonecall: real-world phone call history data (who-calls-whom) containing (sender, receiver, date) triples (e.g. '1234', '5678', '06-DEC-07') in 2007-12-01 to 2008-1-31.
– NELL: real world knowledge base data containing (noun phrase 1, noun phrase 2, context) triples (e.g. 'George Harrison', 'guitars', 'plays') from the 'Read the Web' project [1]. NELL-2 data is the filtered data from NELL by removing entries whose values are below a threshold.
– DARPA1998: 1998 DARPA intrusion detection evaluation dataset from MIT Lincoln Laboratory [20]. We translate the packet data into a 3-way tensor which is composed of (source IP, destination IP, time) triples. For example, a triple ('202.77.162.213', '172.16.114.50', '1998-06-19 08:49:21', '1') corresponds to a packet sent from 202.77.162.213 to 172.16.114.50 at 08:49:21, June 11th, 1998.
– Random: synthetic random tensor of size $I \times I \times I$. The size I varies from 10^3 to 10^8, the number of nonzeros varies from 10^4 to 10^{10}, and the density varies from $10^{-15} \sim 10^{-5}$.

4.2 Scalability

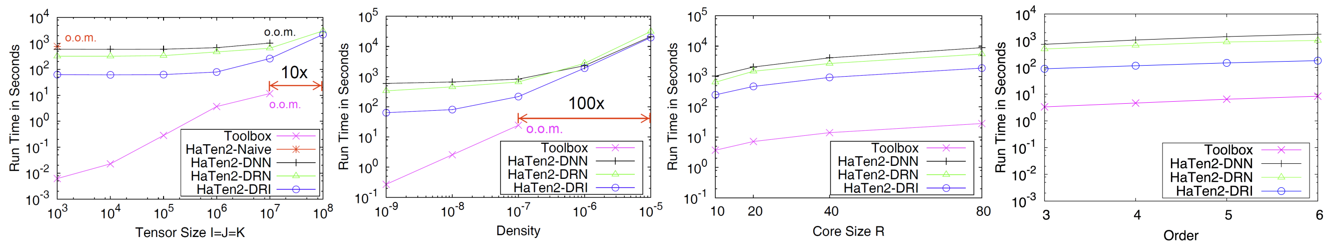
To answer the questions Q1 and Q2, we compare the machine and the data scalabilities of HATEN2-DRI with those of other methods. Since the Tensor Toolbox does not provide nonnegative tucker decomposition, we omit scalability experiments of the Tensor Toolbox.

4.2.1 Data Scalability

Data scalability is measured for the following four aspects: number of nonzeros and dimensionality, density, core tensor size, and order. We change the input tensor in terms of each aspect one by one, while fixing other aspects, and measure the running time using all the 40 machines in the cluster. Since the HATEN2-Naive method cannot process even a 10^4 scale tensor (Figures 1(a) and 7(a)), HATEN2-Naive is omitted from the density and core scalability experiments (Figures 1(b,c) and 7∼ 9(b,c)).

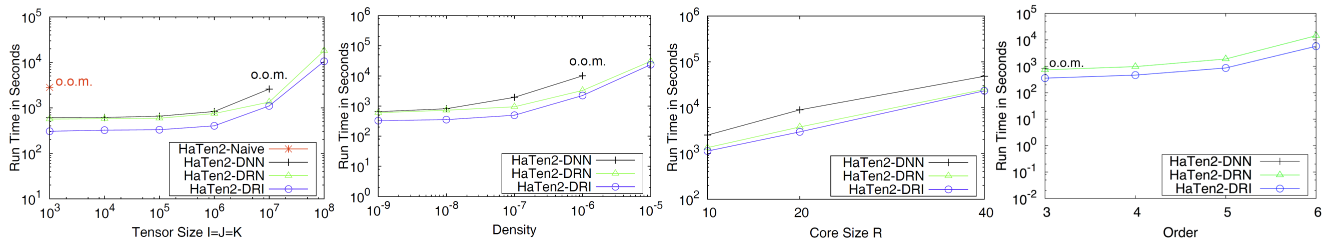
Nonzeros and Dimensionality. We increase the dimensionality $I = J = K$ of modes from 10^3 to 10^8. The number of nonzeros is set to dimensionality $\times 10$. For Tucker decomposition, the size $P \times Q \times R$ of the core tensor is fixed to $10 \times 10 \times 10$. For PARAFAC decomposition, the rank R is set to 10. As shown in Figures 1(a) and 7∼ 9(a), our best method HATEN2-DRI shows the best result: HATEN2-DRI analyzes 10^8 scale tensor the most quickly. HATEN2-Naive and HATEN2-DNN failed for tensors with size beyond 10^3 and 10^7, respectively. Although HATEN2-DRN also analyzes 10^8 scale tensor, the running time is 1.7 times slower than that of HATEN2-DRI.

Density. We increase the density (=number of nonzeros / number of all possible elements) of input tensor from 10^{-9} to 10^{-5}; accordingly, the number of nonzeros becomes 1 billion to 1 trillion, and they take 20MB to 196GB disk space. The dimensionality of each mode is set to 10^5 ($I = J = K$). For Tucker decomposition, the size $P \times Q \times R$ of the core tensor is fixed to $10 \times 10 \times 10$. For PARAFAC decomposition, the rank R is set to 10. As shown in Figures 1(b) and 7∼ 9(b), our method HATEN2-DRI analyzes up to $1000\times$ denser tensors than existing method for nonnegative PARAFAC decomposition and the running time is the fastest compared with other variants of HATEN2.



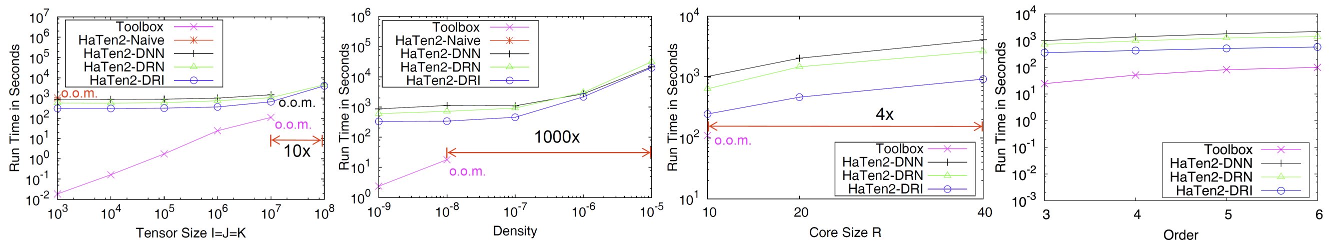
(a) Nonzeros and Dimensionality (b) Density (c) Core (d) Order

Fig. 7: Data scalability of our proposed HATEN2-DRI compared to other methods, for PARAFAC decomposition. The datasets are explained in detail at Section 4.1. o.o.m.: out of memory. Note that our best method HATEN2-DRI decomposes $10 \sim 100\times$ larger data than the Tensor Toolbox.



(a) Nonzeros and Dimensionality (b) Density (c) Core (d) Order

Fig. 8: Data scalability of our proposed HATEN2-DRI compared to other methods, for nonnegative Tucker decomposition. The datasets are explained in detail at Section 4.1. o.o.m.: out of memory. Among the variants of HATEN2, HATEN2-DRI is the fastest, and analyzes $10\times$ denser data than HATEN2-DNN does.



(a) Nonzeros and Dimensionality (b) Density (c) Core (d) Order

Fig. 9: Data scalability of our proposed HATEN2-DRI compared to other methods, for nonnegative PARAFAC decomposition. The datasets are explained in detail at Section 4.1. o.o.m.: out of memory. Note that HATEN2-DRI decomposes $10 \sim 1000\times$ larger data than the Tensor Toolbox and HATEN2 decomposes a tensor with a $4\times$ larger core tensor compared to the Tensor Toolbox. Among the variants of HATEN2, HATEN2-DRI is the fastest, and analyzes $10\times$ larger data than HATEN2-DNN does.

Core Tensor Size. We increase the core size of a random tensor of size $10^6 \times 10^6 \times 10^6$ with 10^7 nonzeros, and measure the running time. For Tucker decomposition, the core tensor size increases from $10 \times 10 \times 10$ to $40 \times 40 \times 40$ or $80 \times 80 \times 80$; for PARAFAC, the rank R increases from 10 to 40 or 80. For normal PARAFAC and Tucker decompositions, as shown in Figures 1(c) and 7(c), HATEN2-DRI scales well, providing the best performance for all the core sizes. When the core size is 80, HATEN2-DRI outperforms the second best method (HATEN2-DRN) by 2.25 times. For nonnegative PARAFAC and Tucker decompositions, as shown in Figures 8(c) and 9(c), HATEN2-DRI scales well up to core size 40. Note that in Figure 9(c), HATEN2-DRI decomposes a tensor with a $4\times$ larger core tensor compared to the Tensor Toolbox.

Order. We increase the order (=number of modes) of a random tensor of size $10^6 \times 10^6 \times 10^6$ with 10^7 nonzeros, and measure the running time. For normal PARAFAC and Tucker decompositions, as shown in Figures 1(d) and 7(d), HATEN2-DRI scales well, providing the best performance among variants of HATEN2 for all the orders. Similar scalability is observed for nonnegative PARAFAC and Tucker decompositions, as shown in Figures 8(d) and 9(d), respectively.

4.2.2 Machine Scalability

To measure the machine scalability, we increase the number of machines from 10 to 40, and report T_{10}/T_M where T_M is the running time with M machines. We use the NELL tensor data of size 26M $\times$26M $\times$48M containing 144M nonzeros. For Tucker and nonnegative Tucker decomposition, the core tensor size is set to $10 \times 10 \times 10$. For PARAFAC and nonnegative PARAFAC decomposition, the rank size is set to 10. As shown in Figure 10, our best method HATEN2-DRI scales near linearly in the beginning, while the performance flattens as the number of machines grows due to overheads required in distributed systems (e.g., synchronization time, JVM loading time, etc.). Note that the machine scalability of Tucker is better than that of PARAFAC. Since Tucker is more complex and involves a lager amount of computations compared with PARAFAC, increasing the number of machines is more effective in Tucker. Due to many sub-operations performed in a local machine, the machine scalabilities of nonnegativity-constrained Tucker and PARAFAC decompositions are smaller than those of the unconstrained versions.

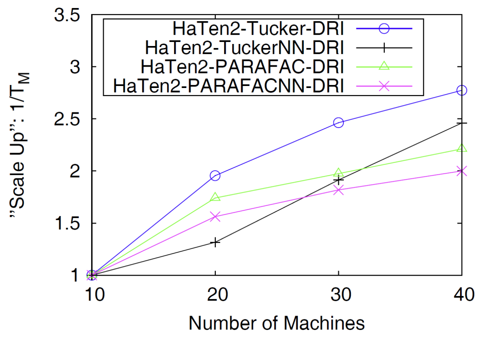


Fig. 10: Machine scalability of HATEN2-PARAFAC-DRI, HATEN2-PARAFACNN-DRI, HATEN2-Tucker-DRI, and HATEN2-TuckerNN-DRI with regard to the "Scale Up" factor $\frac{T_{10}}{T_M}$, where T_M is the running time with M machines. Note that in all cases, HATEN2-DRI scales near linearly in the beginning, while the performance flattens as more machines are added due to overheads in distributed systems.

5 Discovery

In this section, we present discovery results to answer the following questions.

Q1 What are the discoveries on real world tensors?
Q2 What are the differences between PARAFAC and Tucker decompositions on real world tensors?

We apply HATEN2 to four large-scale real-world tensors: Freebase-sampled, DARPA1998, Phonecall, and NELL-2.

5.1 Freebase-sampled

Freebase [19] is a knowledge base dataset in the RDF (Resource Description Framework) format, composed of (subject entity, object entity, relation) triples. Below, we explain construction of a tensor from Freebase and interesting concepts discovered by HATEN2.

Building Freebase-sampled tensor. We build a Freebase-sampled tensor from the full Freebase dataset [19]. First, we extract relations about several important topics including 'Music', 'Books', 'People', 'Film', 'TV', and 'Sports'. Then, we remove the triples containing literal entities (e.g., (John, 'John', name)) since they represent definitions which do not help reveal latent concepts.

Concept discovery. We find several latent concepts in the Freebase-sampled tensor by applying HATEN2-Tucker and HATEN2-PARAFAC on it. We choose the top-k highest valued elements from each column of each factor. Table 8 shows the results by HATEN2-PARAFAC with rank 10. There are several concepts (e.g., 'Pop/Rock Music Albums', 'Expert', and 'Classic') each of which contains groups of subjects, objects, and relations. Note that each subject group is tightly coupled only with the corresponding object and relation groups, due to the diagonal core tensor of PARAFAC. On the other hand, Tucker decomposition gives more diverse concepts determined by cross combinations of various groups. Table 9 shows factors by HATEN2-Tucker with the core size $10 \times 10 \times 10$. We find several groups for each mode: e.g., for the 'subject' mode, we find the groups of 'Instruments', 'Jobs', and 'Marriage'. Table 10 shows concepts each of which combines groups from the subject, the object, and the relation factors. The first concept 'Musician' consists of the subject group S1 ('Instruments'), the object group O1 ('Recording Contributors'), and the relation group R1 ('Professionalism'); the second concept 'Expert' consists of the subject group S2 ('Jobs'), the object group O2 ('Persons'), and the relation group R1 ('Recording Contributors'). Note that the object group R1 appears in both of the concepts, exemplifying the Tucker's ability of finding concepts from various, possibly overlapping groups. The last concept 'Married Couple' consists of the subject group S3 ('Marriage'), the object group O3 ('Spouse'), and the relation group R2 ('Marriage').

Table 8: Concept discovery result from HATEN2-PARAFAC on the Freebase-sampled dataset.

Concepts	Subject Entity	Object Entity	Relation
Concept1: 'Pop/Rock Albums'	Pop Music Alternative rock Rock music	The Day Hell Broke Loose at Sicard Hollow Enigma Variations A Bunch of Stuff The Fifteenth Porn Cut	ns:music.album_release_type.album ns:music.artist.track ns:music.genre.album ns:music.artist.album
Concept2: 'Expert'	Actor Film Producer Comedian Singer	Lia Scott Price Eric Idle Barbra Streisand Madonna	ns:people.profession. people_with_this_profession
Concept3: 'Classic'	Johann Sebastian Bach Pyotr Ilyich Tchaikovsky John Eliot Gardiner Heinrich Schiff	Concerto for Oboe & Violin in D Minor, BWV 1060: II. Adagio Prelude No. 10 in E minor, BWV 855 Choral: "Ach mein herzliebes Jesulein" Cello-Suite No.4 Es-dur BWV 1010: III. Courante	ns:music.artist.album ns:music.artist.track_contribution ns:music.genre.album ns:music.album_release_type.album

Table 9: Discovered factors from HATEN2-Tucker on the Freebase-sampled dataset.

	Subject S1: Instruments	**Subject S2: Jobs**	**Subject S3: Marriage**
Subject Entity	Piano Guitar Electronic Keyboard Bass	Actor Film Score Composer Singer Screenwriter	Marriage
	Object O1: Recording Contributors	**Object O2: Persons**	**Object O3: Spouse**
Object Entity	Pedro Rousseau Yann Tiersen Ari Hest Krischan Frehse	O. Z. Livaneli Eric Idle Jay Chou Haylar Garcia	Helene Belmar Julius & William Safire Beverly McKittrick & Jackie Gleason Kathleen Garman & Jacob Epstein Tyrone Willingham & Kim
	Relation R1: Professionalism	**Relation R2: Marriage**	**Relation R3: Book**
Relation	ns:people.profession .people_with_this_profession	ns:people.marriage_union_type .unions_of_this_type	ns:book.newspaper_issue.publication_date ns:book.poetic_verse_form.poems_of_this_form ns:book.magazine.genre ns:book.book_edition.interior_illustrations_by

Table 10: Concept discovery result from HATEN2-Tucker on the Freebase-sampled dataset.

Concepts	Subject Entity	Object Entity	Relation
Concept1:(S1,O1,R1) 'Musician'	Piano Guitar Electronic Keyboard Bass	Pedro Rousseau Yann Tersen Ari Hest Krischan Frehse	'ns:people.profession .people_with_this_profession'
Concept2: (S2,O2,R1) 'Expert'	Actor Film Score Composer Singer Screenwriter	O. Z. Livaneli Eric Idle Jay Chou Haylar Garcia	'ns:people.profession .people_with_this_profession'
Concept3: (S3,O3,R2) 'Married Couple'	Marriage	Helene Belmar Julius & William Safire Beverly McKittrick & Jackie Gleason Kathleen Garman & Jacob Epstein Tyrone Willingham & Kim	'ns:people.marriage_union_type .unions_of_this_type'

5.2 DARPA1998

DARPA1998 is an intrusion detection evaluation dataset provided by MIT Lincoln Laboratory [20]. They provide packet dump files containing network traffic logs during seven weeks and a list of network-based attacks. The list of attack information contains descriptions of attacks such as source IPs, destination IPs, attack types, and times. Table 11 shows a summary of the attack information. More detailed descriptions are in [20].

Building network traffic tensor. To analyze network traffic logs provided in the DARPA1998 dataset, we convert network packet dump data into a 3-way tensor. First, we extract source and destination IP addresses, and a timestamp for each packet. Next, we map the IP addresses and the timestamps onto natural numbers. Since a timestamp has a con-

Table 11: Descriptions of attacks that occur in the DARPA1998 network traffic logs.

	Description
Attacker IP	135.13.216.191 10.20.30.40 230.1.10.20 194.7.248.53 ...
Victim IP	172.16.112.50 (pascal) 172.16.113.50 (zeno) 172.16.114.50 (marx)
Duration	1998-06-01 ∼ 1998-07-17
Attack type	Denial of Service: back, land, neptune, pod, smurf, syslog, teardrop Sweeping: ip sweep, port sweep Buffer overflow: eject, ffb, format, imap Guessing password: dict, guest etc: ftp-write, loadmodule, nmap, perlmagic, phf ...

Table 12: Network traffic pattern discovery result by HATEN2-PARAFAC on the DARPA1998 dataset.

Traffic Pattern	**Source IP**	**Destination IP**	**Time**	**Description**
Pattern1:'neptune attack'	10.20.30.40	172.16.112.50	1998-07-02, 10:16 A.M.	neptune: Syn flood denial of service on one or more ports.
Pattern2:'Heavy interactions'	135.13.216.191	172.16.112.50	1998-07-01, 09:46 A.M.	There were heavy interactions between 2 machines through telnet.
Pattern3:'Normal traffic'	172.16.112.194 172.16.112.50 172.16.114.148	194.27.251.21 194.7.248.153 197.218.177.69	1998-06-15, 03:27 P.M. 1998-06-30, 11:57 A.M. 1998-07-13, 04:56 P.M.	Normal traffic logs.

Table 13: Discovered factors from HATEN2-Tucker on the DARPA1998 dataset.

	Source IP S1: 'neptune' attacker	**Source IP S2: 'port sweep' attackers**	**Source IP S3: Various attackers**
Source IP	135.13.216.191 10.20.30.40 230.1.10.20	194.7.248.153 194.27.251.21	135.8.60.182 197.182.91.233 197.218.177.69
	Destination IP D1: Victim	**Destination IP D2: Victims**	**Destination IP D3: Attackers**
Destination IP	172.16.112.50	194.7.248.153 172.16.113.50 172.16.112.50	172.16.114.148 194.27.251.21 197.218.177.69
	Time T1: Specific time	**Time T2: Specific time**	**Time T3: Various times**
Time	1998-07-01, 09:46 A.M.	1998-07-09, 12:10 P.M. 1998-07-09, 12:22 P.M. 1998-07-09, 12:22 P.M.	1998-06-11, 2:05 P.M. 1998-06-12, 11:56 A.M. 1998-07-08, 11:16 P.M.

tinuous value, we discretize it into bins of length 0.001 second. Last, we build a 3-way tensor with ('Source IP', 'Destination IP', 'Time') triples where each element in the tensor represents the number of packets for the corresponding triple.

Network traffic pattern discovery. By applying HATEN2 on the tensor constructed above, we discover several interesting patterns summarized in Figure 11. Table 12 shows discovery results by HATEN2-PARAFAC with rank 10. We find several patterns (e.g., 'neptune attack', 'Heavy interaction', and 'Normal traffic') from factor groups that contain source IP, destination IP and time. For each group, we present IP addresses and time that have extraordinarily high scores. In the 'neptune attack' and 'Heavy interaction' patterns, scores are concentrated on certain IP addresses and times since massive packets are exchanged between several machines at a certain point. On the other hand, in the 'Normal traffic' pattern, scores are spread over all IP addresses and times, and the score gaps between entries are small. We apply HATEN2-Tucker to the same tensor; we discover patterns for each factor group and find network traffic patterns by combining the factor groups. Table 13 shows several factors discovered by HATEN2-Tucker with the core size $10 \times 10 \times 10$. For the 'source IP' mode, we find 'neptune attacker', 'port sweep attacker' and 'Various attacker' groups. The 'Various attacker' group performs various types

Table 14: Network traffic pattern discovery result by HATEN2-Tucker on the DARPA1998 dataset.

Traffic Pattern	Source IP	Destination IP	Time	Description
Pattern1:(S1, D1, T1) **'Heavy interaction'**	135.13.216.191 10.20.30.40 230.1.10.20	172.16.112.50	1998-07-01, 09:46 A.M.	There were heavy interactions between 2 machines through telnet.
Pattern2:(S1, D1, T2) **'neptune attack'**	135.13.216.191 10.20.30.40 230.1.10.20	172.16.112.50	1998-07-09, 12:10 P.M. 1998-07-09, 12:22 P.M. 1998-07-09, 12:23 P.M.	neptune: Syn flood denial of service on one or more ports.
Pattern3:(S3, D2, T3) **'Various attack'**	135.8.60.182 197.182.91.233 197.218.177.69	194.7.248.153 172.16.113.50 172.16.112.50	1998-06-11, 2:05 P.M. 1998-06-12, 11:56 A.M. 1998-07-08, 11:16 P.M.	Several attackers perform various types of attacks several times.

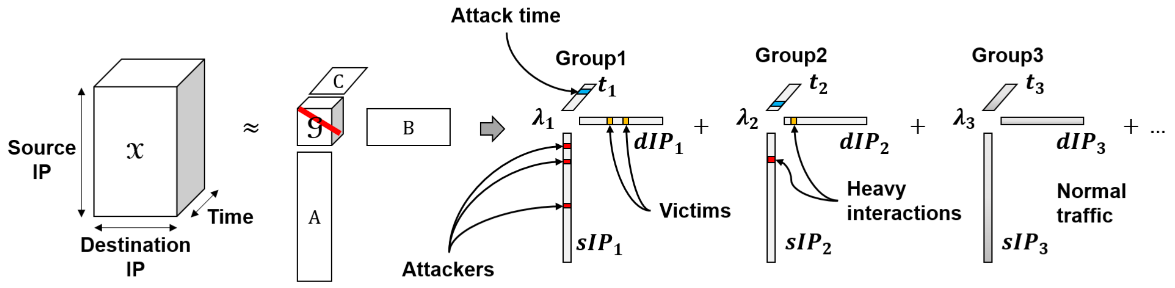


Fig. 11: Network traffic pattern discovery result by HaTen2 on DARPA1998, an intrusion detection dataset. Group1 shows an attack pattern where several attackers perform malicious attacks like denial of service to the victims at a certain point. Group2 shows an interaction pattern where two machines intensively communicate with each other at a certain point. Group3 shows a normal traffic pattern.

of attacks several times. Similarly, we find victim groups and attack time groups from the 'Destination IP' mode and the 'Time' mode, respectively. Since packets are exchanged between attackers and victims, D3 group consists of attackers though it stands for the destination IP address. Table 14 shows the discovered patterns that are combinations of source IP, destination IP and time factor groups. The first group 'Heavy interaction' consists of the source IP group S1 ('neptune attacker'), the destination IP group D1 ("victim"), and the time group T1 ('Specific time'); The second group 'neptune attack' consists of the source IP group S1 ('neptune attacker'), the destination IP group D1 ('victim'), and the time group T2 ('Specific time'). The third group 'Various attack' consists of the source IP group S3 ('Various attacker'), the destination IP group D2 ('victim'), and the time group T3 ('Various time'). The third group shows various attack patterns performed by several attackers at several occasions. Note that the source IP group S1 and the destination IP group D1 are shared by Pattern1 and Pattern2. Since Tucker provides various concepts by cross combinations of factor groups, some factor groups appear in multiple concepts. We also apply the nonnegativity-constrained Tucker and PARAFAC using our HATEN2-TuckerNN and HATEN2-PARAFACNN. The overall result is similar to that of the unconstrained versions of Tucker and PARAFAC. However, the entries that have negatively high scores in the unconstrained versions are scored positively in the nonnegative tensor decompositions. Since all entries have values larger than 0 in nonnegative tensor decomposition, the importance between entities is easily compared. Comparing to the ground truth, we successfully detect attackers and victims in the network traffic logs. For example, we find the following attacker groups: 'neptune' attackers, 'port sweep' attackers, and attackers who perform various types of attacks. Note that we detect the exact attack time for the 'neptune' attack, one of the denial of service attack where attackers send lots of packets to victims intensively at a certain time.

5.3 Phonecall

Phonecall is a real-world phone call history data from an anonymous provider, containing the information of senders, receivers, dates, times, and durations. Below, we explain the construction of a tensor from the Phonecall data, and interesting concepts discovered by HATEN2.

Building Phonecall tensor. To build a 3-way tensor, we extract sender, receiver, and date entries from the Phonecall dataset, and map them to natural numbers. Then, we build a 3-way tensor with ('sender', 'receiver', 'date') triples where

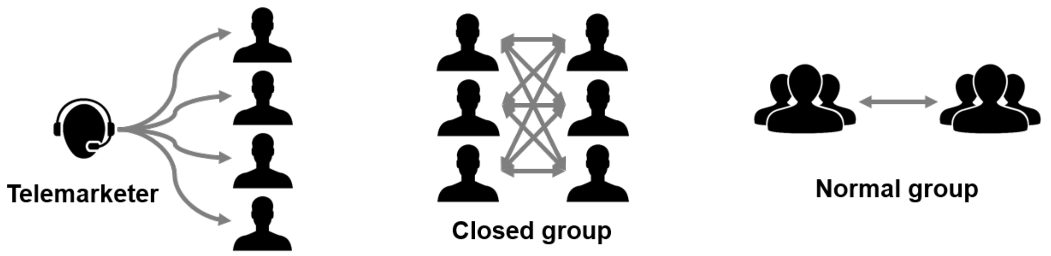


Fig. 12: Discovered sender and receiver groups by HaTen2 on the Phonecall dataset. We find telemarketers who call many people a lot but never receive calls. In the closed group, a large amount of traffics are internally concentrated. There is also a normal group where a person interacts with a small number of people.

Table 15: Discovered factor groups by HaTen2-Tucker on the Phonecall dataset.

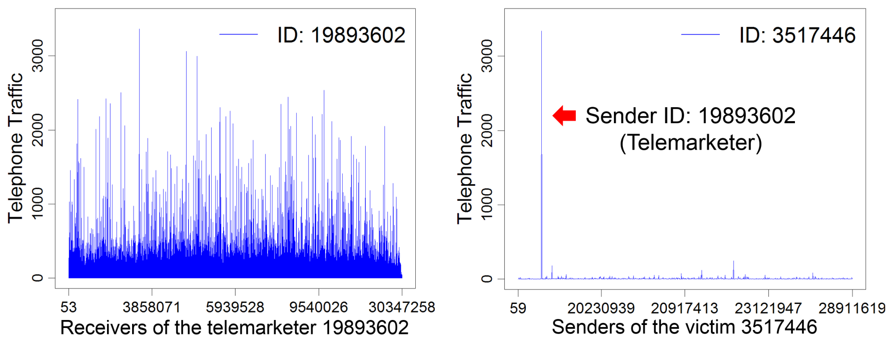
	Sender S1: Telemarketer	**Sender S2: Senders in a closed group**	**Sender S3: Normal**	
Sender_id	19893602 19922918 6657916	5188590 5188591 5188592	7145415 8349492 1944847	
	Receiver R1: Victim	**Receiver R2: Receivers in a closed group**		
Receiver_id	3517446 3517450 4605892	5188590 5188591 5188592		
	Date D1: Normal	**Date D2: Spike**	**Date D3: Periodic**	**Date D4: Christmas**
Date	21-Dec-07 03-Jan-08 04-Jan-08	07-Dec-07	07-Dec-07 09-Dec-07 27-Jan-08	21-Dec-07 22-Dec-07 23-Dec-07

Table 16: Phone call pattern discovery result by HaTen2-Tucker on the Phonecall dataset.

Phone call Pattern	Sender	Receiver	Date
Pattern1:(S1, R1, D1) **'Telemarketing'**	19893602 19922918 6657916	3517446 3517450 4605892	21-Dec-07 03-Jan-08 04-Jan-08
Pattern2:(S2, R2, D3) **'Closed group'**	5188590 5188591 5188592	5188590 5188591 5188592	07-Dec-07 09-Dec-07 27-Jan-08
Pattern3:(S3, R3, D1) **'Normal'**	7145415 8349492 1944847	3517446 3517450 4605892	21-Dec-07 03-Jan-08 04-Jan-08

each element in the tensor represents the number of calls for the corresponding triple.

Phone call pattern discovery. Table 15 shows discovered factor groups after applying HaTen2-Tucker on the Phonecall dataset. We discovered three groups each of which is a combination of factor groups. Figure 12 shows call patterns of those groups, and Table 16 shows in more details how they are composed of. In Table 15, for the sender mode, we find 'Telemarketer', 'Senders in a closed group' and 'Normal' groups. Corresponding to the 'Telemarketer' group, we find the 'Victim' group in the receiver mode. The members of the 'Telemarketer' group call over all the other people, but never receive calls from anyone. Figure 13 shows the telephone traffic pattern of a telemarketer with ID 19893602 and a victim with ID 3517446. Note that the telemarketer calls 218,725 people 46 times on average, and 3,367 times at maximum. The amount of received calls is zero, because they never receive calls from anyone. On the other hand, the members of the 'Victim' group receives many calls from telemarketers, while receiving few calls from normal people. For example, in Figure 13 (b), a receiver with ID 3517446 receives 1 to 10 calls from normal people, but more than 3,000 calls from a telemarketer. These 'Telemarketer' and 'Victim' groups form the 'Telemarketing' group as shown in Table 16. Another group called the 'Closed group' consists of two subgroups: 'Senders in a closed group' for the sender mode and 'Receivers in a closed group' for the receiver mode. In this group, members of the 'Senders in a closed group' send a large amount of traffics to the members of the 'Receivers in a closed group'. The last group we discover is the 'Normal' group. In the 'Normal' group, each member interacts with a small num-



(a) Telemarketer (b) Victim

Fig. 13: Telephone traffics of one of telemarketers with ID 19893602 and one of victims with ID 3517446. The telemarketer calls over all people, but never receives calls from any other. The victim receives lots of calls from the telemarketer while receiving a few calls from normal people.

ber of people in the group. Lastly, we examine the factors for the date mode. Figure 14 shows patterns of the four date mode factors described in Table 15. For the date mode, we find 'Normal', 'Spike', 'Periodic', and 'Christmas' groups. Factor1 ('Normal' factor) shows the normal pattern having similar scores over most days, while having high scores before special events such as Christmas and the New Year's Day. This result makes sense because people usually give words of blessing to their friends right before such special days. Factor2 ('Spike' factor) has an abnormal peak point at 2007-12-07. Factor3 ('Periodic' factor) shows a periodic pattern, and Factor4 ('Christmas' factor) reaches its positive peak right before Christmas and negative peak on 2007-12-07. The 'Normal' factor is contained in the 'Normal' and 'Telemarketing' pattern in Table 16, since telemarketers call many normal people. The 'Periodic' factor is contained in the 'Closed group' pattern since the members in the group interacts with each other periodically.

5.4 NELL-2

NELL is a knowledge base dataset containing ('Noun Phrase 1', 'Noun Phrase 2', 'Context') triples from the 'Read the Web' project [1]. We filter the NELL data by removing entries whose values are below a threshold; the result is a tensor named NELL-2 whose size is $14545 \times 14545 \times 28818$ with 76 millions of nonzeros.

Concept discovery. We discover latent concept groups of NELL-2 by applying HaTen2-PARAFAC with rank 20, and HaTen2-Tucker with the core tensor size $20 \times 20 \times 20$. Table 17 shows the concept discovery results from HaTen2-PARAFAC. We discovered several concepts: e.g., 'Health Care System', 'File Transfer', 'Internet Service', and 'Shopping'. In PARAFAC decomposition, because the core tensor is diagonal, each 'Noun Phrase 1' group is combined only with a 'Noun Phrase 2' group and a 'Context' group. On the other hand, Tucker decomposition provides more diverse concepts compared with PARAFAC decomposition: e.g., a 'Noun Phrase 2' group may be combined with several 'Noun Phrase 1' groups and 'Context' groups. Table 18 shows the groups in factors from Tucker decomposition: e.g., 'Health', 'Credit', 'Network', 'Algorithm', 'Project', and 'Information' in the 'Noun Phrase 1' mode. Table 19 shows the discovered concepts each of which combines the groups from the 'Noun Phrase 1', the 'Noun Phrase 2', and the 'Context' factors. The first concept represents 'Health Care System' which contains the 'Noun Phrase 1' group S1 ('Health'), the 'Noun Phrase 2' group O2 ('Service'), and the 'Context' group C1 ('Care'). Note that a group of a factor appears in several concept groups in Tucker decomposition. For example, the 'Noun Phrase 2' group O2 appears in the first, the second, and the third concepts; the 'Context' group C6 appears in both the second and the third concepts.

Table 17: Concept discovery result using HaTen2-PARAFAC on the NELL-2 dataset.

Concepts	Noun Phrase1	Noun Phrase2	Context
Concept1: 'Health Care System'	health child skin	providers systems organizations	'np1' 'care' 'np2' 'np1' 'insurance' 'np2' 'np1' 'and safety' 'np2'
Concept2: 'File Transfer'	file hypertext FTP	protocol stack technology	'np1' 'stream' 'np2' 'np1' 'transfer' 'np2' 'np2' 'cable' 'np1'
Concept3: 'Internet Service'	internet phone application	providers web sites roots	'np1' 'service' 'np2' 'np1' 'access' 'np2' 'np1' 'hosting' 'np2'
Concept4: 'Shopping'	discount shop grocery	store service products	'np1' 'food' 'np2' 'np1' 'and nutrition' 'np2' 'np1' 'supplement' 'np2'

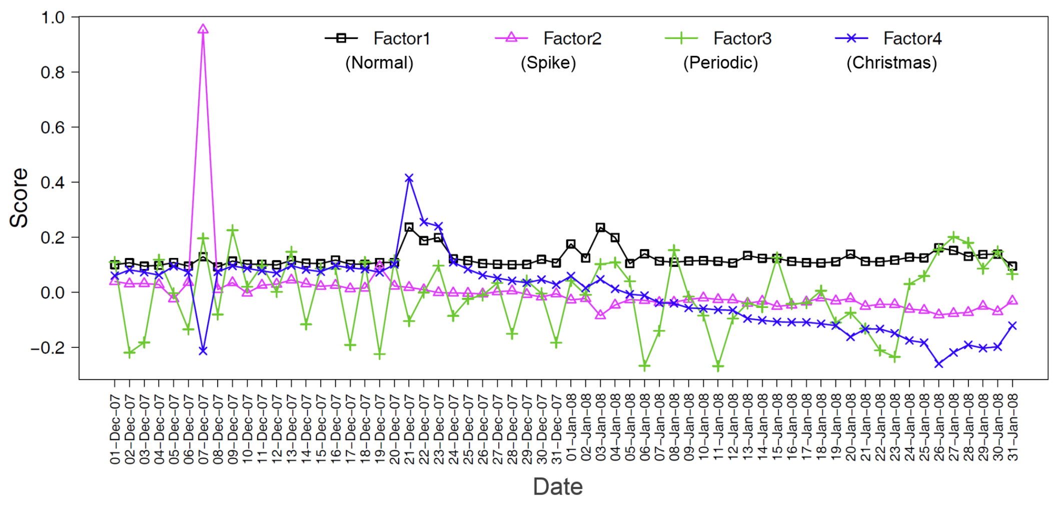


Fig. 14: Discovered patterns for the date factors by HATEN2 on the Phonecall dataset. Factor1 shows a normal pattern that has high scores right before Christmas and the New Year's Day, and remains constant for the other days; Factor2 has an abnormal peak point on 2007-12-07; Factor3 shows a periodic pattern; Factor4 reaches its positive peak right before Christmas and negative peak on 2007-12-07.

Table 18: Discovered factors from HATEN2-Tucker on the NELL-2 dataset.

	NP S1: Health	**NP S2: Credit**	**NP S3: Network**	**NP S4: Algorithm**	**NP S5: Project**	**NP S6: Information**
Noun Phrase1	health child skin eye patient	credit charge bank ID account	internet phone email contact network	optimization rankings listings algorithms indexing	agency proposal management activities manager	information details news material pictures
	NP O1: Region	**NP O2: Service**	**NP O3: Web search**	**NP O4: Research**	**NP O5: Loan**	**NP O6: Network**
Noun Phrase2	world state planet region globe	providers system service insurance organization	search website page industry performance	research experience work training study	loan rates mortgage lender refinancing	roots speeds proxies ports routers
	Context C1: Care	**Context C2: Credit**	**Context C3: Function**	**Context C4: Transfer**	**Context C5: Support**	**Context C6: Service**
Context	'np1' 'care' 'np2' 'np1' 'insurance' 'np2' 'np1' 'service' 'np2' 'np1' 'safety' 'np2' 'np1' 'and fitness' 'np2'	'np1' 'card' 'np2' 'np1' 'report' 'np2' 'np2' 'management' 'np1' 'np1' 'account' 'np2' 'np1' 'debt' 'np2'	'np2' 'engine' 'np1' 'np2' 'returned' 'np1' 'np2' 'results' 'np1' 'np2' 'returns' 'np1' 'np2' 'machine' 'np1'	'np1' 'stream' 'np2' 'np1' 'transfer' 'np2' 'np1' 'communication' 'np2' 'np1' 'protocol' 'np2' 'np2' 'cable' 'np1'	'np2' 'project' 'np1' 'np2' 'and development' 'np1' 'np2 'funding' 'np1' 'np1' 'sponsoring' 'np2' 'np1' 'supporting' 'np2'	'np1' service' 'np2' 'np1 'access np2 'np1' 'hosting' 'np2' 'np1' 'broadband 'np2 'np1' 'infrastructure' 'np2'

6 Related Work

6.1 CP/PARAFAC

Acar et al. [22] use the PARAFAC decomposition in order to detect epilepsy in brain measurements. In [2], Kolda and Bader extend the popular HITS algorithm for ranking webpages, by incorporating anchor text information to the hyperlinks, and using PARAFAC in order to derive hubs and authorities from the data. PARAFAC has also been used in anomaly detection; [3] and [23] detect network anomalies in computer network connection logs and specifically [23] spots anomalies in time-evolving social networks as well. Last but not least, the PARAFAC decomposition has been used in community detection, where we have different views of the same network of people [24], or the network evolves over time and we are interested in identifying communities over time [25].

6.2 Tucker

In [26], apart from a highly memory efficient Tucker decomposition algorithm, there is an overview of the various aspects of the Tucker decomposition as a data mining tool. The authors of [27] use a tensor in order to represent multiple semantic relations (such as "synonym" or "antonym") and use Tucker as a higher order generalization of SVD, in order to perform Latent Semantic Analysis. One of the most

Table 19: Concept discovery result using HATEN2-Tucker on the NELL-2 dataset.

Concepts	Noun Phrase1	Noun Phrase2	Context
Concept1: (S1, O2, C1) 'Health Care System'	health	providers	'np1' 'care' 'np2'
	child	system	'np1' 'insurance' 'np2'
	skin	professionals	'np1' 'service' 'np2'
Concept2: (S3, O2, C6) 'Internet Service'	internet	providers	'np1' 'service' 'np2'
	application	system	'np1' 'access' 'np2'
	email	professionals	'np1' 'hosting' 'np2'
Concept3: (S6, O2, C6) 'Information Access'	information	providers	'np2' 'service' 'np1'
	details	system	'np2' 'access' 'np1'
	news	professionals	'np2' 'hosting' 'np1'
Concept4: (S4, O3, C3) 'Web Search Algorithm'	optimization	search	'np2' 'engine' 'np1'
	rankings	website	'np2' 'returned' 'np1'
	marketing	performance	'np2' 'results' 'np1'
Concept5: (S5, O4, C5) 'Research Project Funding'	agency	research	'np2' 'projects' 'np1'
	grants	training	'np2' 'funding' 'np1'
	proposal	study	'np1' 'sponsoring' 'np2'

widely used Tucker variation is the so called Higher Order Singular Value Decomposition (HOSVD) [28] which is a Tucker3 model with additional orthonormality constraints on the factor matrices. An exemplary work of employing HOSVD is [29] where the authors provide web search recommendations to users. HOSVD has been extensively used in Computer Vision applications [30][31]

6.3 Scalable Algorithms for Tensor Analysis

Bader and Kolda develop efficient algorithms for sparse tensors [32], where they avoid the materialization of very large, unnecessary intermediate Khatri-Rao products. Kang et al. proposed GigaTensor [18] that first uses a distributed system for PARAFAC decomposition. GigaTensor is similar to HATEN2-PARAFAC-DRN in this paper; however in this work we provide a significant improvement upon [18]. It can be shown that the ways that GigaTensor [18] and [32] avoid the intermediate data explosion are equivalent, however, GigaTensor [18] provides an algorithm which is optimized for the distributed setting. In the preliminary version of this paper [10], Jeon et al. unify the large scale Tucker and PARAFAC tensor decomposition algorithms on MAPREDUCE into a general framework, but do not consider the nonnegativity constraint. In [33] Beutel et al. propose FlexiFaCT, a MAPREDUCE algorithm based on Distributed Stochastic Gradient Descent for PARAFAC and coupled PARAFAC decompositions. In [34], Bro and Sidiropoulos use Tucker to compress a tensor, then do the PARAFAC decomposition on the compressed tensor and finally decompress the factors, thus speeding up the PARAFAC decomposition. An alternative approach, DBN, is introduced in [35] where the authors use Relational Algebra to break down the tensor into smaller tensors, using relational decomposition, and thus achieving scalability. Furthermore, [23], introduces ParCube, an approximate and highly paralellizable algorithm for sparse PARAFAC decomposition. For scalable Tucker decomposition, there exists several previous works. Kolda and Sun [26] propose MET (Memory-Efficient Tucker) for scalable Tucker decomposition algorithm running on Matlab. Finally, Erdos and Miettinen introduce a scalable boolean tensor decomposition using random walks [36].

7 Conclusion

In this paper, we propose HATEN2, a distributed method for large-scale tensor decompositions that runs on the MAPREDUCE platform. HATEN2 provides a unified framework to devise efficient MAPREDUCE algorithms for unconstrained and nonnegativity-constrained Tucker and PARAFAC tensor decompositions, which significantly reduces the intermediate data size and the running time. By careful design and implementation, HATEN2 decomposes up to 1000x larger tensors compared to existing methods. Furthermore, HATEN2 scales up near linearly on the number of machines. By applying HATEN2, we discover interesting patterns on various real-world data—knowledge bases, network traffic logs, and phone call history—with millions of rows, columns, and entries which were hard to analyze by existing methods.

Acknowledgments. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning (grants No. 2013005259 and No. 2013R1A1A1064409). The ICT at Seoul National University provides research facilities for this study.

References

1. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *AAAI*, 2010.
2. T. G. Kolda and B. W. Bader, "The tophits model for higher-order web link analysis," in *Workshop on Link Analysis, Counterterrorism and Security*, vol. 7, pp. 26–29, 2006.
3. K. Maruhashi, F. Guo, and C. Faloutsos, "Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis," in *Proceedings of the Third International Conference on Advances in Social Network Analysis and Mining*, 2011.
4. J. Sun, S. Papadimitriou, and P. S. Yu, "Window-based tensor analysis on high-dimensional and multi-aspect streams," in *ICDM*, 2006.
5. T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *ICDM*, pp. 363–372, 2008.
6. I. N. Davidson, S. Gilpin, O. T. Carmichael, and P. B. Walker, "Network discovery via constrained tensor analysis of fmri data.," in *KDD*, pp. 194–202, ACM, 2013.

7. J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: Dynamic tensor analysis," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, (New York, NY, USA), pp. 374–383, ACM, 2006.
8. "Hadoop information." http://hadoop.apache.org/.
9. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *OSDI'04*, Dec. 2004.
10. I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pp. 1047–1058, 2015.
11. R. Harshman, "Foundations of the parafac procedure: model and conditions for an explanatory multi-mode factor analysis," *UCLA working papers in phonetics*, vol. 16, pp. 1–84, 1970.
12. G. Tomasi and R. Bro, "A comparison of algorithms for fitting the parafac model," *Computational Statistics & Data Analysis*, vol. 50, no. 7, pp. 1700–1734, 2006.
13. L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, pp. 279–311, 1966c.
14. C. A. Andersson and R. Bro, "Improving the speed of multi-way algorithms: Part I. Tucker3," *Chemometrics and Intelligent Laboratory Systems*, vol. 42, pp. 93–103, 1998.
15. D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *NIPS*, pp. 556–562, 2000.
16. D. Chen and R. J. Plemmons, "Nonnegativity constraints in numerical analysis," *Symposium on the Birth of Numerical Analysis*, 2007.
17. Y. D. Kim and S. Choi, "Nonnegative tucker decomposition," in *CVPR*, IEEE Computer Society, 2007.
18. U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *KDD*, pp. 316–324, 2012.
19. "Freebase dataset." https://www.freebase.com/.
20. "Darpa 1998 dataset." http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1998data.html.
21. B. W. Bader, T. G. Kolda, *et al.*, "Matlab tensor toolbox version 2.5," January 2012.
22. E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener, "Multiway analysis of epilepsy tensors," *Bioinformatics*, vol. 23, no. 13, pp. i10–i18, 2007.
23. E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *Machine Learning and Knowledge Discovery in Databases*, pp. 521–536, Springer, 2012.
24. E. E. Papalexakis, L. Akoglu, and D. Ienco, "Do more views of a graph help? community detection and clustering in multi-graphs," in *Information Fusion (FUSION), 2013 16th International Conference on*, pp. 899–905, IEEE, 2013.
25. M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra, "Com2: Fast automatic discovery of temporal (comet) communities," in *Advances in Knowledge Discovery and Data Mining*, pp. 271–283, Springer, 2014.
26. T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *ICDM 2008: Proceedings of the 8th IEEE International Conference on Data Mining*, pp. 363–372, 2008.
27. K. W. Chang, W. T. Yih, and C. Meek, "Multi-relational latent semantic analysis.," in *EMNLP*, pp. 1602–1612, 2013.
28. L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.
29. J. Sun, H. Zeng, H. Liu, Y. Lu, and Z. Chen, "Cubesvd: a novel approach to personalized web search," in *WWW*, 2005.
30. M. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," *Computer Vision ECCV 2002*, pp. 447–460, 2002.
31. D. Luo, H. Huang, and C. Ding, "Discriminative high order svd: Adaptive tensor subspace selection for image classification, clustering, and retrieval," in *ICCV*, 2011.
32. B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, pp. 205–231, December 2007.
33. A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on hadoop," in *SDM*, 2014.
34. R. Bro, N. Sidiropoulos, and G. Giannakis, "A fast least squares algorithm for separating trilinear mixtures," in *Int. Workshop Independent Component and Blind Signal Separation Anal*, pp. 11–15, 1999.
35. M. Kim and K. S. Candan, "Decomposition-by-normalization (dbn): leveraging approximate functional dependencies for efficient tensor decomposition," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 355–364, ACM, 2012.
36. D. Erdös and P. Miettinen, "Scalable boolean tensor factorizations using random walks," *CoRR*, vol. abs/1310.4843, 2013.