Chapter 7
Effective Tensor-Based Data Clustering Through Sub-Tensor Impact Graphs

K. Selçuk Candan, Shengyu Huang, Xinsheng Li, and Maria Luisa Sapino

7.1 Introduction

Tensors are multi-dimensional arrays and are commonly used for representing multi-dimensional data, such as sensor streams and social networks [9, 17]. Thanks to the widespread availability of multi-dimensional data, tensor decomposition operations (such as CP [10] and Tucker [31]) are increasingly being used to implement various data analysis tasks, from anomaly detection [17], correlation analysis [26] to pattern discovery [13] and clustering [22, 28, 32].

A critical challenge for tensor-based analysis is its computational complexity and decomposition can be a bottleneck in some applications [14, 21, 30]. Phan and Cichocki [23] proposed a methodology to partition the tensor into smaller sub-tensors to deal with this issue: (a) partition the given tensor into blocks (or sub-tensors), (b) decompose each block independently, and then (c) iteratively combine these sub-tensor decompositions into a final decomposition for the input tensor. This process leads to two key observations:

- Observation #1: Our key observation in this chapter is that *Step (c), which iteratively updates and stitches the sub-tensor decompositions obtained in Steps (a) and (b), is where the various decompositions interact with each other and*

K. S. Candan (✉) · S. Huang · X. Li
Arizona State University, Tempe, AZ, USA
e-mail: candan@asu.edu; shengyu.huang@asu.edu; lxinshen@asu.edu

M. L. Sapino
University of Torino, Torino, Italy
e-mail: marialuisa.sapino@unito.it

O. Nasraoui, C.-E. Ben N'Cir (eds.), *Clustering Methods for Big Data Analytics*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-319-97864-2_7

where any inaccuracies in individual sub-tensor decompositions can propagate (through the update rules introduced in Sect. 7.2.4) to the decomposition of the complete tensor.

- Observation #2: We further observe that *if we can quantify and capture how these sub-tensors interact and inaccuracies propagate, we can use this information to better allocate resources to tackle the accuracy–efficiency trade-off inherent in the decomposition process.*

Based on these two observations, in this chapter, we introduce the notion of *sub-tensor impact graphs (SIGs)* (Sect. 7.3), which capture and represent how the decompositions of these sub-partitions impact each other and the overall tensor decomposition accuracy and present several complementary algorithms that leverage this novel concept to address various key challenges in tensor decomposition.

7.1.1 Contributions of This Chapter: Sub-Tensor Impact Graphs

While block-based tensor decomposition techniques [15, 23] provide potential opportunities to boost the accuracy/efficiency trade-off, this solution leaves several open questions, including (a) how to partition the tensor and (b) how to most effectively combine results from these partitions. In this chapter, we introduce the notion of *sub-tensor impact graphs (SIGs)*, which quantify how the decompositions of these sub-partitions impact each other and the overall tensor decomposition accuracy and present four complementary algorithms that leverage this novel concept to address various key challenges in tensor decomposition, including *personalization*, *noise*, and *dynamic data*.

7.1.1.1 Challenge #1: Decomposition in the Presence of Dynamic Data

Firstly, we rely on sub-tensor impact graphs (SIGs) to tackle performance challenges that dynamic data pose in tensor analytics: incremental tensor decomposition. Recomputation of the whole tensor decomposition with each update will cause high computational costs and incur large memory overheads. Especially for applications where data evolves over time and the tensor-based analysis results need to be continuously maintained. In Sect. 7.4, we present a two-phase block-incremental CP-based tensor decomposition technique (BICP), which relies on sub-tensor impact graphs to prune unnecessary computation in the presence of incremental updates on the data [11].

7.1.1.2 Challenge #2: Dealing with Noisy Data

Next, in Sect. 7.5, we present a *Noise Adaptive Tensor Decomposition (*`nTD`*)* method that leverages sub-tensor impact graphs to tackle deal with noisy data. `nTD` partitions the tensor into multiple sub-tensors and then decomposes each sub-tensor probabilistically through Bayesian factorization—the resulting decompositions are then recombined through an iterative refinement process to obtain the decomposition for the whole tensor. `nTD` leverages a resource allocation strategy that accounts for the impact of the noise density of one sub-tensor on the decomposition accuracies of the other sub-tensors, based on the underlying sub-tensor impact graph [19].

7.1.1.3 Challenge #3: Personalization of the Decomposition Process

Finally, we introduce a novel personalized tensor decomposition (PTD) mechanism for accounting for the user's focus and interests during tensor decomposition (Sect. 7.6). We present alternative ways to account for the impact of the accuracy of one region of the tensor to the accuracies of the other regions of the tensor, each based on a different assumption about how the impact of inaccuracies propagates along the tensor. Given a model of impact, PTD (a) first partitions the input tensor in a way that reflects user's interest, (b) constructs a sub-tensor impact graph reflecting the tensor content and its partitions, and then (c) analyzes this sub-tensor impact graph (in the light of the user's interest) to identify initial decomposition ranks for the sub-tensors in a way that will boost the final decomposition accuracies for partitions of interest [18].

7.2 Background

7.2.1 Tensors

A tensor is a multi-dimensional array. More formally, an N-way or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system. A third-order tensor has three indices. A first-order tensor is a vector, a second-order tensor is a matrix, and tensors of order three or higher are called higher-order tensors. As in the case of matrices, the dimensions of the tensor array are referred to as its modes. For example, the tensor, $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, shown in Fig. 7.1, is of third-order and has three modes: I columns (mode 1), J rows (mode 2), and K tubes (mode 3). Fibers are the higher-order analogue of matrix rows and columns. A fiber is defined by fixing every index but one. A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. Slices are two-dimensional sections of a tensor, defined by fixing all but two indices [16].

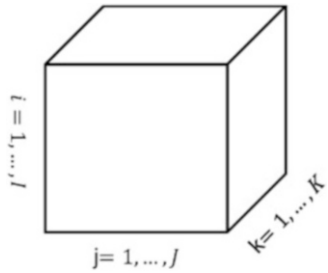


Fig. 7.1 A third-order (3-mode) tensor of dimensions, $I \times J \times K$

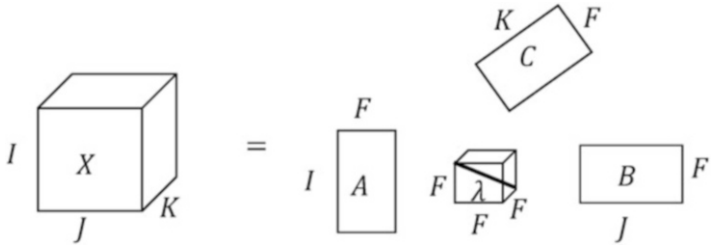


Fig. 7.2 CP decomposition of a 3-mode tensor results in a diagonal core and three factors

7.2.2 Tensor Decomposition

Tensor-based algorithms, most notably tensor decomposition, are increasingly important tools for analysis, including clustering, of high-dimensional data sets. Intuitively, tensor decomposition process generalizes matrix decomposition-based data analysis and clustering (such as PCA [7] and SVD [5, 8]) to high-dimensional arrays (known as tensors) and rewrites the given tensor in the form of a set of factor matrices (one for each mode of the input tensor) and a core tensor (which, intuitively, describes the spectral structure of the given tensor). These factor matrices and core tensors then can be used for obtaining multi-modal clusters of the input data. The two most popular tensor decomposition algorithms are the Tucker [31] and the CANDECOMP/PARAFAC(CP) [10] decompositions. We next provide a brief description of these algorithms.

7.2.2.1 CP and Tucker Decompositions

The PARAFAC decomposition can be seen as a generalization of matrix factorizations to tensors [10]. PARAFAC decomposition is also known as CANDECOMP/PARAFAC (CP) decomposition. As shown in Fig. 7.2, given a tensor $\mathcal{X}$, CP factorizes the tensor into F component matrices (where F is a user supplied non-zero integer value also referred to as the *rank* of the decomposition). For the simplicity of the discussion, let us consider a 3-mode tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$. CP would decompose $\mathcal{X}$ into $\mathring{\mathcal{X}}$ consisting of three matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, such that

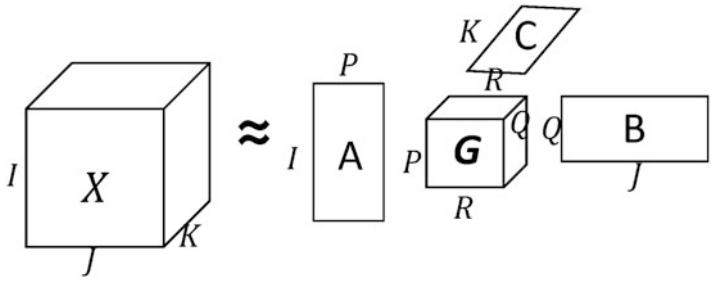


Fig. 7.3 Tucker decomposition of a three-mode tensor

$$\mathcal{X} \approx \tilde{\mathcal{X}} = recombine[\mathring{\mathcal{X}}] \equiv recombine[\mathbf{A}, \mathbf{B}, \mathbf{C}] \equiv \sum_{f=1}^{F} a_f \circ b_f \circ c_f,$$

where $a_f \in \mathbb{R}^I$, $b_f \in \mathbb{R}^J$, and $c_f \in \mathbb{R}^K$. The factor matrices **A**, **B**, **C** are the combinations of the rank-one component vectors into matrices, e.g., $\mathbf{A} = [\, a_1\, a_2 \cdots a_F \,]$. This is visualized in Fig. 7.2.

Tucker decomposition generalizes singular value matrix decomposition (SVD) to higher-dimensional data (Fig. 7.3). Given a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, Tucker decomposition factorizes the tensor into factor matrices with different number of rows, which are referred to as the rank of the decomposition. Tucker decomposition would decompose $\mathcal{X}$ into three matrices **A**, **B**, **C** and one core dense tensor **G**, such that

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \mathbf{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \equiv \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r,$$

where $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$ are the factor matrices and can be treated as the principal components in each mode. The (dense) core tensor, $\mathbf{G} \in \mathbb{R}^{P \times Q \times R}$, indicates the strength of interactions among different components of the factor matrices.

7.2.2.2 Accuracy of Tensor Decomposition

Note that, in general, unlike matrix decomposition (where each matrix has an exact decomposition), tensors may not have exact decompositions [16]. Therefore, many of the algorithms for decomposing tensors are based on an iterative process that tries to improve the approximation until a convergence condition is reached, such as an alternating least squares (ALS) method: at its most basic form, ALS estimates, at each iteration, one factor matrix while maintaining other matrices fixed; this process is repeated for each factor matrix associated with the modes of the input tensor.

Note that due to the approximate nature of tensor decomposition operation, given a decomposition [**A**, **B**, **C**] of $\mathcal{X}$, the tensor $\tilde{\mathcal{X}}$ that one would obtain by re-composing the tensor by combining the factor matrices **A**, **B**, and **C** is often different from the input tensor, $\mathcal{X}$. The accuracy of the decomposition is often measured by considering the Frobenius norm of the difference tensor:

$$accuracy(\mathcal{X}, \tilde{\mathcal{X}}) = 1 - error(\mathcal{X}, \tilde{\mathcal{X}}) = 1 - \left(\frac{\|\tilde{\mathcal{X}} - \mathcal{X}\|}{\|\mathcal{X}\|} \right).$$

7.2.3 Tensor Decomposition and Clustering

As we mentioned earlier, intuitively, tensor decomposition process generalizes matrix decomposition to high-dimensional arrays and the resulting factor matrices and core tensors then can be used for obtaining multi-modal clusters of the input data. Indeed, tensor-based representations of data and tensor decompositions (especially the two widely used decompositions CP [10] and Tucker [31]) are proven to be effective in multi-aspect data analysis and clustering. For instance, [22] used tensor decomposition to cluster patients in a health-care setting based on their individual and health profile data, including age, medical history, and diagnostics: in particular, the authors have created a patient information tensor and decomposed this tensor (by nonnegative low-rank approximation methods) to obtain semantic clusters that can be used to characterize patients' records. Davidson et al. [6] applied tensor decomposition to fMRI data to help differentiating healthy and Alzheimer affected individuals. Cao et al. [3] used a similar tensor decomposition-based approach to cluster face images: authors modeled a collection of faces as a tensor and they applied a tensor-based principal component analysis for seeking face clusters. Wu et al. [32] leveraged CP decomposition (solved through stochastic gradient descent) to cluster heterogeneous information networks: each type of object in the network is represented as a different mode of the tensor. Sun et al. [28], on the other hand, has shown that Tucker decomposition can be used for subspace clustering which simultaneously conducts dimensionality reduction and membership representation.

7.2.4 Block-Based Tensor Decomposition

One key challenge with tensor decomposition is its computational complexity: decomposition algorithms have high computational costs and, in particular, incur large memory overheads (also known as the *intermediary data blow-up problem*) and, thus, basic algorithms and naive implementations are not suitable for large problems. HaTen2 [12] focuses on sparse tensors and presents a scalable tensor

Algorithm 1 The outline of the block-based iterative improvement process

Input: original tensor $\mathcal{X}$, partitioning pattern $\mathcal{K}$, and decomposition rank, F
Output: CP tensor decomposition $\breve{\mathcal{X}}$

1. Phase 1: for all $\mathbf{k} \in \mathcal{K}$
   - decompose $\mathcal{X}_{\mathbf{k}}$ into $U_{\mathbf{k}}^{(1)}, U_{\mathbf{k}}^{(2)}, \ldots, U_{\mathbf{k}}^{(N)}$
2. Phase 2: repeat
   a. for each mode $i = 1$ to N
      i. for each modal partition, $k_i = 1$ to K_i,
         A. update $A_{(k_i)}^{(i)}$ using $U_{[*,\ldots,*,k_i,*,\ldots,*]}^{(i)}$, for each block $\mathcal{X}_{[*,\ldots,*,k_i,*,\ldots,*]}$; more specifically,
            - compute $T_{(k_i)}^{(i)}$, which involves the use of $U_{[*,\ldots,*,k_i,*,\ldots,*]}^{(i)}$ (i.e., the mode-i factors of $\mathcal{X}_{[*,\ldots,*,k_i,*,\ldots,*]}$)
            - revise $P_{[*,\ldots,*,k_i,*,\ldots,*]}$ using $U_{[*,\ldots,*,k_i,*,\ldots,*]}^{(i)}$ and $A_{(k_i)}^{(i)}$
            - compute $S_{(k_i)}^{(i)}$ using the above
            - update $A_{(k_i)}^{(i)}$ using the above
            - for each $\mathbf{k} = [*, *, \ldots, k_i, \ldots, *, *]$
              – update $P_{\mathbf{k}}$ and $Q_{\mathbf{k}}$ using
              – $U_{\mathbf{k}}^{(i)}$ and $A_{(k_i)}^{(i)}$

   until stopping condition
3. Return $\breve{\mathcal{X}}$

decomposition suite of methods for Tucker and PARAFAC decompositions on the MapReduce framework. TensorDB [15] leverages a chunk-based framework to store and retrieve data, extends array operations to tensor operations, and introduces optimization schemes for in-database tensor decomposition.

One way to deal with this challenge is to partition the tensor and obtain the tensor decomposition leveraging these smaller partitions. Block-based decomposition techniques partition the given tensor into blocks or sub-tensors, initially decompose each block independently, and then iteratively combine these decompositions into a final decomposition. GridPARAFAC [23], for example, partitions the tensor into pieces, obtains decomposition for each piece (potentially in parallel), and stitches the partial decomposition results into a combined decomposition for the initial tensor through an iterative improvement process. Here, we provide an overview of the block-based tensor decomposition process.

Let us consider an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$ where $\mathcal{K}$ is the set of sub-tensor indexes. Without loss of generality, let us assume that $\mathcal{K}$ partitions the mode i into K_i equal partitions, i.e., $|\mathcal{K}| = \prod_{i=1}^{N} K_i$. Let us also assume that we are given a target decomposition rank, F, for the tensor $\mathcal{X}$. Let us further assume that each sub-tensor in $\mathfrak{X}$ has already been decomposed with target rank F and let $\mathfrak{U}^{(i)} = \{U_{\mathbf{k}}^{(i)} \mid \mathbf{k} \in \mathcal{K}\}$

denote the set of F-rank sub-factors[1] corresponding to the sub-tensors in $\mathfrak{X}$ along mode i. In other words, for each $\mathcal{X}_{\mathbf{k}}$, we have

$$\mathcal{X}_{\mathbf{k}} \approx I \times_1 U^{(1)}_{\mathbf{k}} \times_2 U^{(2)}_{\mathbf{k}} \cdots \times_N U^{(N)}_{\mathbf{k}}, \tag{7.1}$$

where I is the N-mode $F \times F \times \ldots \times F$ identity tensor, where the diagonal entries are all 1s and the rest are all 0s. Given these, Phan and Cichocki [23] presents an iterative improvement algorithm for composing these initial sub-factors into the full F-rank factors, $A^{(i)}$ (each one along one mode), for the input tensor, $\mathcal{X}$. The outline of this block- based process is as follows: Let us partition each factor $A^{(i)}$ into K_i parts corresponding to the block boundaries along mode i:

$$A^{(i)} = [A^{(i)T}_{(1)} A^{(i)T}_{(2)} ... A^{(i)T}_{(K_i)}]^T.$$

Given this partitioning, each sub-tensor $\mathcal{X}_{\mathbf{k}}$, $\mathbf{k} = [k_1, \ldots, k_i, \ldots, k_N] \in \mathcal{K}$ can be described in terms of these sub-factors:

$$\mathcal{X}_{\mathbf{k}} \approx I \times_1 A^{(1)}_{(k_1)} \times_2 A^{(2)}_{(k_2)} \cdots \times_N A^{(N)}_{(k_N)} \tag{7.2}$$

Moreover [23] shows that the current estimate of the sub-factor $A^{(i)}_{(k_i)}$ can be revised using the update rule (for more details on the update rules please see [23]):

$$A^{(i)}_{(k_i)} \longleftarrow \quad T^{(i)}_{(k_i)} \left(S^{(i)}_{(k_i)}\right)^{-1} \tag{7.3}$$

where

$$T^{(i)}_{(k_i)} = \sum_{\mathbf{l} \in \{[*,\ldots,*,k_i,*,\ldots,*]\}} U^{(i)}_{\mathbf{l}} \left(P_{\mathbf{l}} \oslash (U^{(i)T}_{\mathbf{l}} A^{(i)}_{(k_i)})\right)$$

$$S^{(i)}_{(k_i)} = \sum_{\mathbf{l} \in \{[*,\ldots,*,k_i,*,\ldots,*]\}} Q_{\mathbf{l}} \oslash \left(A^{(i)T}_{(k_i)} A^{(i)}_{(k_i)}\right)$$

such that, given $\mathbf{l} = [l_1, l_2, \ldots, l_N]$, we have

- $P_{\mathbf{l}} = \circledast_{h=1}^{N} (U^{(h)T}_{\mathbf{l}} A^{(h)}_{(l_h)})$ and $Q_{\mathbf{l}} = \circledast_{h=1}^{N} (A^{(h)T}_{(l_h)} A^{(h)}_{(l_h)})$.

Above, $\circledast$ denotes the Hadamard product and $\oslash$ denotes element-wise division.

The block-based tensor decomposition process is outlined in pseudocode in Algorithm 1. Figure 7.4 provides a visual example of this process: The given input tensor $\mathcal{X}$ is partitioned into two sub-tensors, $\mathcal{X}_1$ and $\mathcal{X}_2$. In the first stage, each sub-tensor is decomposed by CP, thus obtaining partial factors. The second stage

[1] If the sub-tensor is empty, then the factors are 0 matrices of the appropriate size.

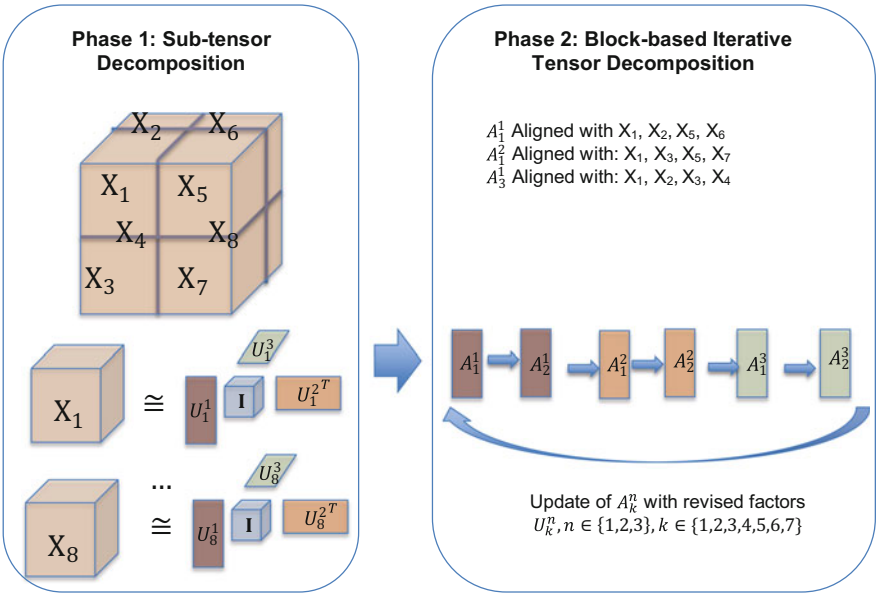


Fig. 7.4 Illustration of block-based tensor decomposition process

combines these partial decomposed factors using iterative updates to derive the final factors (and the corresponding core) for tensor $\mathcal{X}$.

7.3 Sub-Tensor Impact Graphs (SIGs) and Sub-Tensor Impact Scores

In this section, we formally introduce the concept of *sub-tensor impact graph (SIG)* that captures and represents the underlying structure of sub-tensors and helps efficiently calculate the impact of each sub-tensor on the decomposition accuracy of the overall tensor.

Let an N-mode tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, be partitioned into a grid, $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$, of sub-tensors, such that

- K_i indicates the number of partitions along mode-i,
- the size of the jth partition along mode i is $I_{j,i}$ (i.e., $\sum_{j=1}^{K_i} I_{j,i} = I_i$), and
- $\mathcal{K} = \{[k_{j_1}, \ldots, k_{j_i}, \ldots, k_{j_N}] \mid 1 \leq i \leq N,\ 1 \leq j_i \leq K_i\}$ is a set of sub-tensor indexes.

The number, $\|\mathfrak{X}\|$, of partitions (and thus also the number, $\|\mathcal{K}\|$, of partition indexes) is $\prod_{i=1}^{N} K_i$.

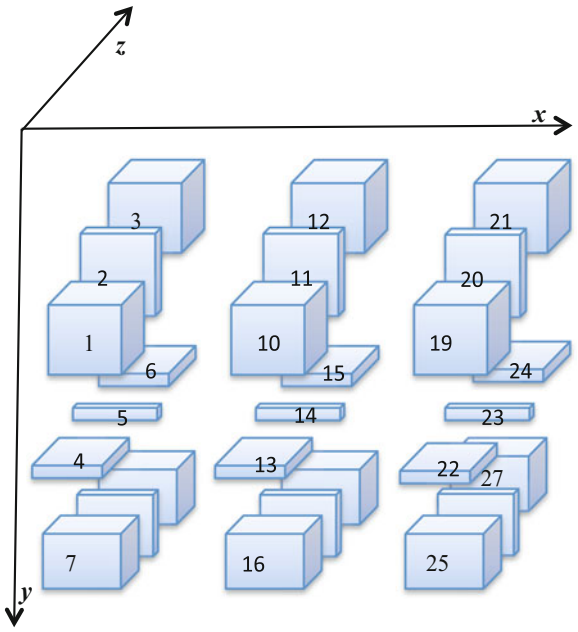


Fig. 7.5 A sample 3-mode tensor, partitioned into 27 heterogeneous sub-tensors

Example 7.1 Figure 7.5 shows a 3-mode tensor, partitioned into 27 sub-tensors: 12 tensor-blocks (sub-tensors 1, 3, 7, 9, 10, 12, 16, 18, 19, 21, 15, 27), 12 slices (sub-tensors 2, 8, 11, 17, 20, 26, 4, 6, 13, 15, 22, 24), and three fibers (sub-tensors 5, 14, 23). The specific shapes of partitions may correspond to user's requirement such as the degree of importance or user focus.

7.3.1 Accuracy Dependency Among Sub-Tensors

In Sect. 7.2.4, we presented update rules block-based tensor decomposition algorithms use for stitching the individual sub-tensor decompositions into a complete decomposition for the whole tensor. While the precise derivation of these update rules is not critical for our discussion (and is beyond the scope of this chapter), it is **important** to note that, as visualized in Fig. 7.6, each $A^{(i)}_{(k_i)}$ is maintained incrementally by using, for all $1 \leq j \leq N$, the current estimates for $A^{(j)}_{(k_j)}$ and the decompositions in $\mathfrak{U}^{(j)}$, i.e., the F-rank sub-factors of the sub-tensors in $\mathcal{X}$ along the different modes of the tensor. Moreover, and most importantly for the present discussion, this update rule for $A^{(i)}_{(k_i)}$ supports the following observation: Given

$$\mathcal{X}_{\mathbf{k}} \approx \mathcal{I} \times_1 A^{(1)}_{(k_1)} \times_2 A^{(2)}_{(k_2)} \cdots \times_N A^{(N)}_{(k_N)},$$

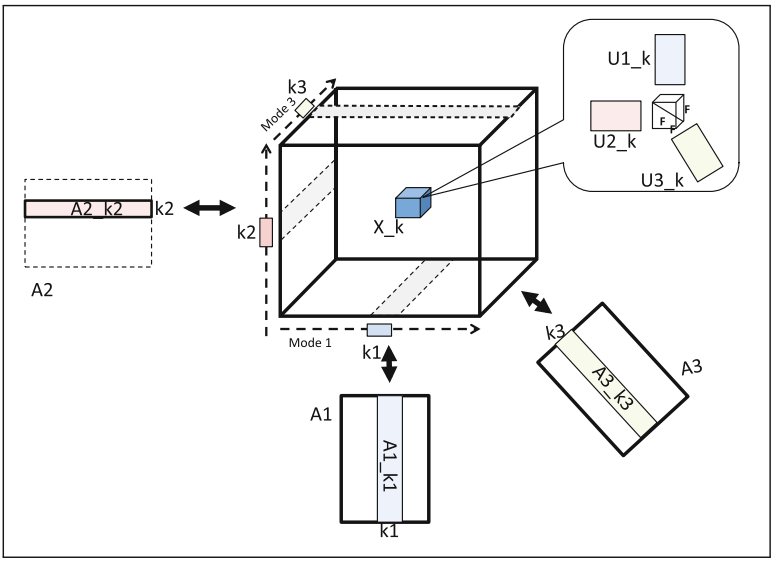


Fig. 7.6 The block-based update rule maintains $A^{(i)}_{(k_i)}$ incrementally by using the current estimates for $A^{(j)}_{(k_j)}$ and the decompositions in $\mathfrak{U}^{(j)}$

the final accuracy for the sub-tensor $\mathcal{X}_{\mathbf{k}}$, $\mathbf{k} = [k_1, \ldots, k_i, \ldots, k_N]$, depends on the accuracies of sub-factors $A^{(i)}_{(k_i)}$. Moreover, the accuracy of each of these, in turn, depends on the accuracies of the sub-factors of the contributing sub-tensors. More specifically, when updating $A^{(i)}_{(k_i)}$, we need to compute

- $T^{(i)}_{(k_i)}$, which involves the use of $U^{(i)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ (i.e., the mode-i factors of $\mathcal{X}_{[*,\ldots,*,k_i,*,\ldots,*]}$), and
- $P_{[*,\ldots,*,k_i,*,\ldots,*]}$, which in turn uses $U^{(h)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ for $1 \leq h \leq N$ (i.e., all factors of $\mathcal{X}_{[*,\ldots,*,k_i,*,\ldots,*]}$).

Therefore, the final accuracy of $\mathcal{X}_{\mathbf{k}}$ depends directly on the initial decomposition accuracies of the factor matrices $U^{(h)}_{[*,\ldots,*,k_j,*,\ldots,*]}$, for $1 \leq i, h \leq N$.

In other words, for each sub-tensor $\mathcal{X}_{\mathbf{k}}$, there is a set, $direct\_impact(\mathcal{X}_{\mathbf{k}}) \subseteq \mathfrak{X}$, of sub-tensors that consists of those sub-tensors whose initial decomposition accuracies directly impact the final decomposition accuracy of $\mathcal{X}_{\mathbf{k}}$. Moreover, as visualized in Fig. 7.7, $direct\_impact(\mathcal{X}_{\mathbf{k}})$ consists of those sub-tensors that are aligned (i.e., share the same slices) with $\mathcal{X}_{\mathbf{k}}$, along the different modes of the tensor.

7.3.2 Sub-Tensor Impact Graphs (SIGs)

Given the accuracy dependencies among the sub-tensors formalized above, we can define a *sub-tensor impact graph (SIG)*:

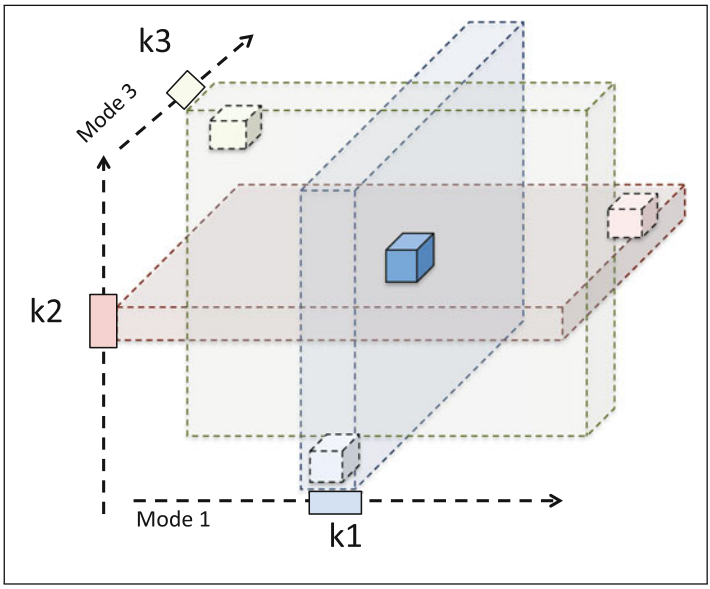


Fig. 7.7 The sub-tensors whose initial decomposition accuracies directly impact given sub-tensors are aligned (i.e., share the same slices) with that sub-tensor along the different modes of the tensor

Definition 7.1 Let an N-mode tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, be partitioned into a grid, $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$, of sub-tensors. The corresponding *sub-tensor impact graph (SIG)* is a directed, weighted graph, $G(V, E, w())$, where

- for each $\mathcal{X}_{\mathbf{k}} \in \mathcal{X}$, there exists a corresponding $v_{\mathbf{k}} \in V$,
- for each $\mathcal{X}_{\mathbf{l}} \in direct\_impact(\mathcal{X}_{\mathbf{k}})$, there exists a directed edge $v_{\mathbf{l}} \rightarrow v_{\mathbf{k}}$ in E, and
- $w()$ is an edge weight function, such that $w(v_{\mathbf{l}} \rightarrow v_{\mathbf{k}})$ quantifies the direct accuracy impact of decomposition accuracy of $\mathcal{X}_{\mathbf{l}}$ on $\mathcal{X}_{\mathbf{k}}$. ◊

Intuitively, the sub-tensor impact graph represents how the decomposition accuracies of the given set of sub-tensors of an input tensor impact the overall combined decomposition accuracy. A key requirement, of course, is to define the edge weight function, $w()$, that quantifies the accuracy impacts of the sub-tensors that are related through update rules. In this section, we introduce three alternative strategies to account for the propagation of impacts within the tensor during the decomposition process.

7.3.2.1 Alt. #1: Uniform Edge Weights

The most straightforward way to set the weights of the edges in E is to assume that the propagation of the inaccuracies over the sub-tensor impact graph is uniform. In other words, in this case, for all $e \in E$, we set $w_{\text{uni}}(e) = 1$.

7.3.2.2 Alt. #2: Surface of Interaction-Based Weights

While being simple, the uniform edge weight alternative may not properly account for the impact of the varying dimensions of the sub-tensors on the error propagation.

As we see in Fig. 7.5, in general, the neighbors of a given sub-tensor can be of varying shape and dimensions and we may need to account for this diversity in order to properly assess how inaccuracies propagate in the tensor. In particular, in this subsection, we argue that the *surface of interaction* between two sub-tensors $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$, defined as below, may need to be considered to account for impact propagation:

Definition 7.2 (Surface of Interaction) Let $\mathcal{X}$ be a tensor partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$. Let also $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$ be two sub-tensors in $\mathfrak{X}$, such that

- $\mathbf{j} = [k_{j_1}, k_{j_2}, \ldots, k_{j_N}]$ and
- $\mathbf{l} = [k_{l_1}, k_{l_2}, \ldots, k_{l_N}]$.

We define the *surface of interaction*, $surf(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}})$, between $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$ as follows:

$$surf(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}}) = \prod_{h \ s.t. \ j_h = l_h} I_{j_h,h}.$$

◊

Here $I_{j_h,h}$ is the size of the j_hth partition along mode h.

Principle 1 *Let $G(V, E, w())$ be a sub-tensor impact graph and let $(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) \in E$ be an edge in the graph. The weight of this edge from $v_{\mathbf{j}}$ to $v_{\mathbf{l}}$ should reflect the area of the surface of interaction between the sub-tensors $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$.*

Intuitively, this principle verbalizes the observation that impacts are likely to propagate more easily if two sub-tensors share large dimensions along the modes on which their partitions coincide. Under this principle, we can set the weight of the edge $(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) \in E$ as follows:

$$w_{\text{sur}}(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) = \frac{surf(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}})}{\sum_{(v_{\mathbf{j}} \rightarrow v_{\mathbf{m}}) \in E} surf(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{m}})}.$$

7.3.2.3 Alt. #3: Value Alignment-Based Edge Weights

Although surface of interaction-based edge weights can potentially account for the varying shapes and sizes of the sub-tensors of $\mathcal{X}$, they fail to take into account for how similar these sub-tensors are—more specifically, they ignore how the values within the sub-tensors are distributed and whether these distributions are aligned across them.

Intuitively, if the value distributions are aligned (or similar) along the modes that two sub-tensors share, then they are likely to have high impacts on each other's decomposition during the decomposition process. If they are dissimilar, on the other hand, their impacts on each other will be minimal. Therefore, considering only the area of the surface of interaction may not be sufficient to properly account for the inaccuracy propagation within the tensor. More specifically, we need to measure the value alignment between sub-tensors as well:

Definition 7.3 (Value Alignment) Let $\mathcal{X}$ be a tensor partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$. Let also $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$ be two sub-tensors in $\mathfrak{X}$, such that

- $\mathbf{j} = [k_{j_1}, k_{j_2}, \ldots, k_{j_N}]$ and
- $\mathbf{l} = [k_{l_1}, k_{l_2}, \ldots, k_{l_N}]$.

Let, $A = \{h \mid k_{j_h} = k_{l_h}\}$ be the set of modes along which the two sub-tensors are aligned and let R be the remaining modes. We define the *value alignment*, $align(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}}, A)$, between $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$ as

$$align(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}}, A) = cos(\mathbf{c}_{\mathbf{j}}(A), \mathbf{c}_{\mathbf{l}}(A)),$$

where the vector $\mathbf{c}_{\mathbf{j}}(A)$ is constructed from the sub-tensor $\mathcal{X}_{\mathbf{j}}$ as follows[2]:

$$\mathbf{c}_{\mathbf{j}}(A) = vectorize(\mathcal{M}_{\mathbf{j}}(A))$$

and the tensor $\mathcal{M}_{\mathbf{j}}(A)$ is constructed from $\mathcal{X}_{\mathbf{j}}$ by fixing the values along the modes in A: $\forall 1 \leq i_h \leq I_{j_h,h}$,

$$\mathcal{M}_{\mathbf{j}}(A)[i_1, i_2, \ldots, i_{|A|}] = norm(\mathcal{X}_{\mathbf{j}}|_{A,i_1,i_2,\ldots,i_{|A|}}).$$

Here, $norm()$ is the standard Frobenius norm and $\mathcal{X}_{\mathbf{j}}|_{A,i_1,i_2,\ldots,i_{|A|}}$ denotes the part of $\mathcal{X}_{\mathbf{j}}$ where the modes in A take values i_1,i_2, through $i_{|A|}$. ◊

Intuitively, $\mathbf{c}_{\mathbf{j}}(A)$ captures the value distribution of the tensor $\mathcal{X}_{\mathbf{j}}$ along the modes in A.

Principle 2 *Let $G(V, E, w())$ be a sub-tensor impact graph and let $(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) \in E$ be an edge in the graph. The weight of this edge from $v_{\mathbf{j}}$ to $v_{\mathbf{l}}$ should reflect the structural alignment between the sub-tensors $\mathcal{X}_{\mathbf{j}}$ and $\mathcal{X}_{\mathbf{l}}$.*

This principle verbalizes the observation that impacts are likely to propagate more easily if two given sub-tensors are structurally aligned along the modes on which their partitions coincide. As before, under this principle, we can set the edge weights of the edge $(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) \in E$ in the sub-tensor impact graph as follows:

$$w_{\text{align}}(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) = \frac{align(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}})}{\sum_{(v_{\mathbf{j}} \rightarrow v_{\mathbf{m}}) \in E} align(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{m}})}.$$

[2] $\mathbf{c}_{\mathbf{l}}(A)$ is similarly constructed from sub-tensor $\mathcal{X}_{\mathbf{l}}$.

7.3.2.4 Alt. #4: Combined Edge Weights

The surface of interaction-based edge weights account for the shapes of the sub-tensors, but do not account for their value alignments. In contrast, value alignment-based edge weights consider the structural similarities of the sub-tensors, but ignore how big the surfaces they share are.

Therefore, a potentially more effective alternative would be to combine these surface of interaction and value alignment-based edge weights into a single weight that takes into account both aspects of sub-tensor interaction:

$$w_{\text{comb}}(v_{\mathbf{j}} \rightarrow v_{\mathbf{l}}) = \frac{comb(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{l}})}{\sum_{(v_{\mathbf{j}} \rightarrow v_{\mathbf{m}}) \in E} comb(\mathcal{X}_{\mathbf{j}}, \mathcal{X}_{\mathbf{m}})},$$

where $comb(\mathcal{Y}, \mathcal{Z}) = align(\mathcal{Y}, \mathcal{Z}) \times surf(\mathcal{Y}, \mathcal{Z})$.

7.3.3 Sub-Tensor Impact Scores

While the edges on the sub-tensor impact graph, G, account for how (in)accuracies propagate during each individual application of the update rules, it is important to note that after several iterations of updates, *indirect* propagation of impacts also occur over the graph G:

- during the first application of the update rule, impacts propagate among the sub-tensors that are immediate neighbors;
- during the second application of the update rule, impacts reach from one sub-tensor to those sub-tensors that are 2-hop away;

  ...
- during the mth application of the rule, impacts propagate to the m-hop neighbors of each sub-tensor.

In order to use the sub-tensor impact graph to assign resources, we therefore need to measure how impacts propagate within G over a large number of iterations of the alternating least squares (ALS) process.

For this purpose, we rely on a random-walk-based measure of node relatedness on the given graph. More specifically, we rely on personalized PageRank (PPR [2, 4]) to measure sub-tensor relatedness. Like all random-walk-based techniques, PPR encodes the structure of the graph in the form of a transition matrix of a stochastic process and complements this with a seed node set, $S \subseteq V$, which serves as the context in which scores are assigned: each node, v_i in the graph is associated with a score based on its positions in the graph relative to this seed set (i.e., how many paths there are between v_i and the seed set and how short these paths are). Intuitively, these seeds represent sub-tensors that are critical in the given application (e.g. high-update, high-noise, or high-user-relevance; see Sects. 7.3 through 7.3.2 for various applications).

Given the graph and the seeds, the PPR score $\mathbf{p}[i]$ of v_i is obtained by solving the following equation:

$$\mathbf{p} = (1 - \beta)\mathbf{T}_G\ \mathbf{p} + \beta\mathbf{s},$$

where $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\mathbf{s}$ is a re-seeding vector such that if $v_i \in S$, then $\mathbf{s}[i] = \frac{1}{\|S\|}$ and $\mathbf{s}[i] = 0$, otherwise. Intuitively, $\mathbf{p}$ is the stationary distribution of a random walk on G which follows graph edges (according to the transition probabilities $\mathbf{T}_G$) with probability $(1 - \beta)$ and jumps to one of the seeds with probability β. Correspondingly, those nodes that are close to the seed nodes over a large number of paths obtain large scores, whereas those that are poorly connected to the nodes in S receive small PPR scores. We note that the iterative nature of the random-walk process underlying PPR fits well with how inaccuracies propagate during the iterative ALS process. Based on this observation, given a directed, weighted *sub-tensor impact graph (SIG)*, $G(V, E, w())$, we construct a transition matrix, $\mathbf{T}_G$, and obtain the PPR score vector $\mathbf{p}$ by solving the above equation.[3] The resulting *sub-tensor impact scores* are then used for assigning appropriate resources to the various sub-tensors as described in the next three sections.

7.4 Application #1: Block-Incremental CP Decomposition (BICP) and Update Scheduling Based on Sub-Tensor Impact Scores

There are many applications in which data is evolving dynamically. Obviously, in such scenarios, re-computation of the whole tensor decomposition with each update will cause high computational costs. In this section, we present a block-incremental CP decomposition (BICP) scheme which leverages SIGs to efficiently conduct the iterative refinement process during the second phase of the block-based tensor decomposition process. Let us assume that we are given a tensor, $\mathcal{X}$, with decomposition, $\mathring{\mathcal{X}}$, and an update, Δ, on the tensor. BICP significantly reduces computational cost of obtaining the decomposition of the updated tensor, while maintaining high accuracy by relying on two complementary techniques:

- **Update-Sensitive Block Maintenance in First Phase:** In its first phase of the process, instead of repeatedly conducting ALS on each sub-tensor, BICP only revises the decompositions of the sub-tensors that contain updated data. Moreover, when the update is small with respect to the block size, BICP relies on incremental factor tracking [20, 27] to avoid re-decomposition of the updated sub-tensor.

[3] Note that, since in general, the number of partitions is small and is independent of the size of the input tensor, the cost of the PPR computation to obtain the ranks is negligible next to the cost of tensor decomposition.

- **Update-Sensitive Refinement in the Second Phase:** In its second phase, BICP leverages (automatically extracted) metadata about how decompositions of the sub-tensors impact each other's decompositions and a block-centric iterative refinement to help achieve high efficiency and accuracy:
  - BICP limits the refinement process to only those blocks that are aligned with the updated block.
  - We employ sub-tensor impact graph (SIG) to account for the refinement relationships among the sub-tensors; we further apply impact score to reduce redundant work: we
    - identify sub-tensors that do not need to be refined and (probabilistically) prune them from further consideration, and/or
    - assign different ranks to different sub-tensors according to their impact score: naturally, the larger the impact likelihood of a sub-tensor is, the larger target rank BICP assigns to that tensor.

    Intuitively, the above process enables BICP to assign appropriate levels of accuracy to sub-tensors in a way that reflects the distribution of the updates on the whole tensor. This ensures that the process is fast and accurate.

In this chapter, we focus on the SIG-based update sensitive refinement during the second phase of the block-based decomposition process.

7.4.1 Reducing Redundant Refinements

During the refinement process of Phase 2, those sub-tensors that have direct refinement relationships with the updated sub-tensors are critical to the refinement process. Our key observation is that if we could quantify how much an update on a sub-tensor impacts sub-factors on other sub-tensors, then we could use this to optimize Phase 2. More specifically, given an update, Δ on tensor $\mathcal{X}$, BICP assigns an update sensitive impact score, $I_{\Delta}(\mathcal{X}_{\mathbf{k}})$ to each sub-tensor, $\mathcal{X}_{\mathbf{k}}$, and leverages this impact score to regulate the refinement process to eliminate redundant work

Intuitively, if the two sub-tensors are similarly distributed along the modes that they share, then they are likely to have high impacts on each other's decomposition; Therefore we use alternative #3: value alignment-based edge weights to assign the weight of edge (introduced in Sect. 7.3.2). To calculate an update sensitive impact score, we can rely on personalized PageRank (introduced in Sect. 7.3.3) to measure sub-tensor relatedness. PPR encodes the structure of the graph in the form of a transition matrix of a stochastic process from which the significances of the nodes in the graph can be inferred. Here, we choose updated sub-tensors as seed nodes and calculate PPR scores for all the other nodes as their impact scores.

- *Optimization Phase 2-I:* Intuitively, if a sub-tensor has a low impact score, its decomposition is minimally affected given the update, Δ. Therefore, those sub-tensors with very low-impact factors can be completely ignored in the refinement process and their sub-factors can be left as they are without any refinement.
- *Optimization Phase 2-P:* While optimization phase 2-I can potentially save a lot of redundant work, completely ignoring low-impact tensors may have a significant impact on accuracy. An alternative approach, with a less drastic impact than ignoring sub-tensors, is to associate a refinement probability to sub-tensors based on their impact scores. In particular, instead of completely ignoring those sub-tensors with low-impact factors, we assign them an update probability, $0 < prob\_update < 1$. Consequently, while the factors of sub-tensors with high impact scores are refined at every iteration of the refinement process, factors of sub-tensors with low-impact scores have lesser probabilities of refinement and, thus, do not get refined at every iteration of Phase 2.
- *Optimization Phase 2-R:* A second alternative to completely ignoring the refinement process for low-impact sub-tensors is to assign different ranks to different sub-tensors according to their impact scores: naturally, the higher the target rank is, the more accurate the decomposition of the sub-tensor is. We achieve this by adjusting the decomposition rank, $F_{\mathbf{k}}$ of $\mathcal{X}_{\mathbf{k}}$, as a function of the corresponding tensor's update sensitive impact score:

$$F_{\mathbf{k}} = \left\lceil F \times \frac{I_\delta(\mathcal{X}_{\mathbf{k}})}{max_{\mathbf{h}}\{I_\delta(\mathcal{X}_{\mathbf{h}})\}} \right\rceil.$$

  Intuitively, this formula sets the decomposition rank of the sub-tensor with the highest impact score relative to the given update, Δ, to F; other sub-tensors are assigned progressively smaller ranks (potentially all the way down to 1)[4] based on their impacts scores. Once the new ranks are computed, we obtain new $U_{(k)}$ factors with partial ranks $F_{\mathbf{k}}$ for $\mathcal{X}_{\mathbf{k}}$ and refine these incrementally in Phase 2.

  Here, we consider two rank-based optimization strategies, *phase 2-Ra* and *phase 2-Ri*. In *phase 2-Ra*, we potentially adjust the decomposition rank for all relevant sub-tensors. In *phase 2-Ri*, however, we adjust ranks only for sub-tensors with high impact on the overall decomposition.

By extending the complexity formulation from [23], we can obtain the complexity[5] of Phase 2 as $O((F \times \sum_{i=1}^{N} \frac{I_i}{K_i} + F^2) \times \mathcal{T} \times H \times |\mathcal{D}|)$ where $\mathcal{T}$ is the number of refinement iterations, $H = (100 - L)\%$ is the ratio of high impact sub-tensors maintained, $|\mathcal{D}|$ is the number of sub-tensors that have direct impact on updated sub-tensors, I_i is the dimensionality of the tensor along mode i, and K_i is the number of partitions along that mode.

[4] It is trivial to modify this equation such that the smallest rank will correspond to a user provided lower bound, $F_{\min}$, when such a lower bound is provided by the user.

[5] Here we report the complexity of $phase2 - I$ and other refinement method complexity can be derived similarly.

7.4.2 Evaluation

In this section, we report sample results that aim to assess the effectiveness of the proposed BICP approach in helping eliminate redundant refinements

7.4.2.1 Setup

Data Sets In these experiments, we used three data sets: *Epinions* [29], *Ciao* [29], and *Enron* [24]. The first two of these are comparable in terms of their sizes and semantics: they are both $5000 \times 5000 \times 27$ tensors, with schema $\langle user, item, category \rangle$, and densities 1.089×10^{-6} and 1.06×10^{-6}, respectively. The *Enron* email data set, on the other hand, has dimensions $5632 \times 184 \times 184$, density 1.8×10^{-4}, and schema $\langle time, from, to \rangle$.

Data Updates We divided the tensor into 64 blocks (using $4 \times 4 \times 4$ partitioning) and applied all the updates to four of these blocks; Once the blocks are selected, we randomly pick a slice on the block and update 10% of the fibers on this slice.

Alternative Strategies We consider the following strategies to maintain the tensor decomposition: Firstly, we apply the basic two-phase block-centric decomposition strategy, i.e., we decompose all sub-tensors with CPALS in Phase 1 and we apply iterative refinement using all sub-tensors in Phase 2 (in the charts, we refer to this non-incremental decomposition approach as `ORI`). For Phase 1, we use a version of STA where we update fibers that are update-critical, i.e., with highest energy among all the affected fibers. For Phase 2, again, we have several alternatives: (a) applying Phase 2 without any impact score-based optimization (`P2N`), (b) ignoring $L\%$ of sub-tensors with the lowest impact scores (`P2I`), (c) reducing the decomposition rank of sub-tensors (`P2Ra` and `P2Ri`), or (d) using probabilistic refinements for sub-tensors with low impact scores (`P2P`). In these experiments, we choose $L = 50\%$ and, for `P2P`, we set the update probability to $p = 0.1$. In addition to the block-based BCIP and its optimizations, we also considered, as an efficient alternative, application of the incremental factor tracking process to the whole tensor as in STA [27]—in the charts, we refer to this alternative approach as `Whole`.

Evaluation Criteria We use the measure reported in Sect. 7.2.2.1 to assess decomposition accuracy. We also report decomposition time for different settings. In these experiments, the target decomposition rank is set to $F = 10$. Unless otherwise specified, the maximum number of iterations in Phase 2 is set to 1000. Each experiment was run 100 times and averages are reported.

Hardware and Software We used a quad-core Intel(R) Core(TM)i5-2400 CPU @ 3.10GHz machine with 8.00GB RAM. All codes were implemented in Matlab and run using Matlab 7.11.0 (2010b) and Tensor Toolbox Version 2.5 [1].

7.4.2.2 Discussion of the Results

Impact scores measure how different regions of the tensor impacts other parts of the tensor during the alternating least squares (ALS) process. Therefore, we expect that, when we leverage the impact scores (computed in a way to account for the distribution of the data updates) to assign the decomposition ranks, we should be able to focus the decomposition work to better fit the dynamically evolving data. Figure 7.8 compares execution times and accuracies of several approaches. Here,

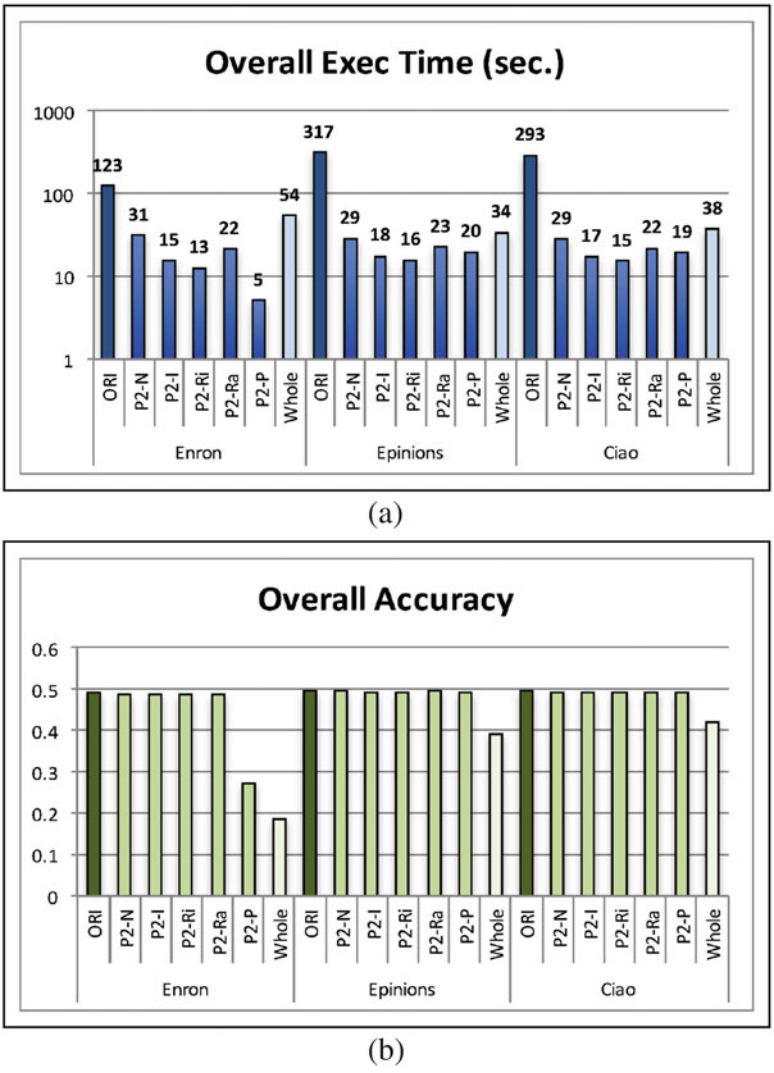


Fig. 7.8 Comparison of (**a**) Execution times and (**b**) Decomposition accuracies under the default configuration: the proposed optimizations provide several orders of gain in execution time relative to `ORI`, while (unlike `Whole`) they match `ORI`'s accuracy

ORI indicates non-incremental two-phase block centric decomposition, whereas Whole indicates application of factor tracking to the whole tensor. The other five techniques in the figure (P2N, P2I, P2Ri, P2Ra, P2P) all correspond to optimizations of the proposed BICP approach for Phase 2.

Firstly, this figure shows that the two social media data sets, Epinions and Ciao, with similar sizes and densities show very similar execution time and accuracy patterns. The figure also shows that the Enron data set also exhibits a pattern roughly similar to the other data sets, despite having a different size and density.

The key observation in Fig. 7.8 is that the SIG-based optimizations provide several orders of gain in execution time while matching the accuracy of non-optimized version almost perfectly (i.e., the optimizations come without accuracy penalties). In contrast, the alternative strategy, Whole, which incrementally maintains the factors of the whole tensor (as opposed to maintaining the factors of its blocks) also provides execution time gains, but sees a significant drop in its accuracy.

We note that P2P, which probabilistically updates low-impact sub-tensors rather than completely ignoring them, does not significantly improve accuracy. This is because the P2I approach already has an accuracy almost identical to P2N, i.e., ignoring low-impact tensors is a very safe and effective method to save redundant work. Therefore, also considering that, unless a large number of blocks are ignored, P2I is able to match the accuracy of P2N, we do not see a major need to use P2P to reduce the impact of ignored sub-tensors.

7.5 Application #2: Noise-Profile Adaptive Decomposition (nTD) and Sample Assignment Based on Sub-Tensor Impact Scores

Many of the tensor decomposition schemes are sensitive to noisy data, an inevitable problem in the real world that can lead to false conclusions. Recent research has shown that it is possible to avoid over-fitting by relying on probabilistic techniques that leverage Gibbs sampling-based Bayesian model learning [33]; however, these assume that all the data and intermediary results can fit in the main memory, and (b) they treat the entire tensor uniformly, ignoring potential non-uniformities in the noise distribution. In this chapter, we present a *Noise Adaptive Tensor Decomposition (*nTD*)* method, which leverages a probabilistic two-phase decomposition strategy, complemented with sub-tensor impact graphs, to develop a sample assignment strategy that best suits the noise distribution of the given tensor to leverage available *rough* knowledge regarding where in the tensor noise might be more prevalent.

Algorithm 2 Phase 1: Monte Carlo-based Bayesian decomposition of each sub-tensor

Input: Sub-tensor $\mathcal{X}_{\mathbf{k}}$, sampling number L

Output: Decomposed factors $U_{\mathbf{k}}^{(1)}, U_{\mathbf{k}}^{(2)}, \ldots, U_{\mathbf{k}}^{(N)}$

1. Initialize model parameters $U_{\mathbf{k}}^{(1)1}, U_{\mathbf{k}}^{(2)1}, \ldots, U_{\mathbf{k}}^{(N)1}$.
2. For $l = 1, \ldots, L$
   a. Sample the hyper-parameter, α:
      - $\alpha^l \sim p(\alpha^l | U_{\mathbf{k}}^{(1)l}, U_{\mathbf{k}}^{(2)l}, \ldots, U_{\mathbf{k}}^{(N)l}, \mathcal{X}_{\mathbf{k}})$
   b. For each mode $j = 1, \ldots, N$,
      i. Sample the corresponding hyper-parameter, Θ:
         - $\Theta_{U_{\mathbf{k}}^{(j)l}} \sim p(\Theta_{U_{\mathbf{k}}^{(j)l}} | U_{\mathbf{k}}^{(j)l})$
      ii. For $i_j = 1, ..., I_j$, sample the mode (in parallel):

$$U_{\mathbf{k}(i_j)}^{(j)(l+1)} \sim p\left(U_{\mathbf{k}(i_j)}^{(j)} \middle| U_{\mathbf{k}}^{(1)l}, \ldots, U_{\mathbf{k}}^{(j-1)l}, U_{\mathbf{k}}^{(j+1)l}, \ldots, U_{\mathbf{k}}^{(N)l}, \Theta_{U_{\mathbf{k}}^{(j)}}^{l}, \alpha^l, \mathcal{X}_{\mathbf{k}}\right)$$

3. For each mode $j = 1, \ldots, N$,
   - $U_{\mathbf{k}}^{(j)} = \frac{\sum_{i=1}^{L} U_{\mathbf{k}}^{(j)i}}{L}$

More specifically, nTD partitions the tensor into multiple sub-tensors and then decomposes each sub-tensor probabilistically through Bayesian factorization—the resulting decompositions are then recombined through an iterative refinement process to obtain the decomposition for the whole tensor. We refer to this as *Grid-Based Probabilistic Tensor Decomposition* (GPTD). nTD complements GPTD with a SIG-based resource allocation strategy that accounts for the impact of the noise density of one sub-tensor on the decomposition accuracies of the other sub-tensors. This provides several benefits: Firstly, the partitioning helps ensure that the memory footprint of the decomposition is kept low. Secondly, the probabilistic framework used in the first phase ensures that the decomposition is robust to the presence of noise in the sub-tensors. Thirdly, a priori knowledge about noise distribution among the sub-tensors is used to obtain a resource assignment strategy that best suits the noise profile of the given tensor.

7.5.1 Grid-Based Probabilistic Tensor Decomposition (GPTD)

As a block-based algorithm, *Grid-Based Probabilistic Tensor Decomposition* (GPTD) partitions the given tensor into blocks, decomposes each block independently, and then iteratively combines these decompositions into a final composition. Differently from Algorithm 1, however, GPTD leverages Monte Carlo-based

Bayesian decomposition of sub-tensors in its Phase 1 (see Algorithm 2) to better deal with the problem of over-fitting, which is a challenge especially when the data is noisy.

Intuitively, entries in the factor matrices are modeled as probabilistic variables and decomposition is posed as a maximization problem where these (latent) random variables fit the observed data. In the presence of noise in the data, the observed variables may also be modeled probabilistically: since the observations cannot be precisely described, they may be considered as samples from a probability distribution. In this section, following [25], in the presence of data uncertainty (due to noise), we describe the fit between the observed data and the predicted latent factor matrices, probabilistically, as follows:

$$\mathcal{X}_{\mathbf{k}(i_1,i_2,\ldots,i_N)} \Big| U_{\mathbf{k}}^{(1)}, U_{\mathbf{k}}^{(2)} \ldots, U_{\mathbf{k}}^{(N)} \sim \mathcal{N}([U_{\mathbf{k}(i_1)}^{(1)}, U_{\mathbf{k}(i_2)}^{(2)} \ldots, U_{\mathbf{k}(i_N)}^{(N)}], \alpha^{-1}), \tag{7.4}$$

where the conditional distribution of $\mathcal{X}_{\mathbf{k}(i_1,i_2,\ldots,i_N)}$ given $U_{\mathbf{k}}^{(j)}$ $(1 \leq j \leq N)$ is a Gaussian distribution with mean $[U_{\mathbf{k}(i_1)}^{(1)}, U_{\mathbf{k}(i_2)}^{(2)}, \ldots, U_{\mathbf{k}(i_N)}^{(N)}]$ and the observation precision α. We also impose independent Gaussian priors on the modes:

$$U_{\mathbf{k}(i_j)}^{(j)} \sim \mathcal{N}(\mu_{U_{\mathbf{k}}^{(j)}}, \Lambda_{U_{\mathbf{k}}^{(j)}}^{-1}) \quad i_j = 1 \ldots I_j \tag{7.5}$$

where I_j is the dimensionality of the jth mode. Given this, one can estimate the latent features $U_{\mathbf{k}}^{(j)}$ by maximizing the logarithm of the posterior distribution, $\log p(U_{\mathbf{k}}^{(1)}, U_{\mathbf{k}}^{(2)} \ldots, U_{\mathbf{k}}^{(N)} | \mathcal{X}_{\mathbf{k}})$.

One difficulty with the approach, however, is the tuning of the hyper-parameters of the model: α and $\Theta_{U_{\mathbf{k}}^{(j)}} \equiv \{\mu_{U_{\mathbf{k}}^{(j)}}, \Lambda_{U_{\mathbf{k}}^{(j)}}\}$ for $1 \leq j \leq N$. [33] notes that one can avoid the difficulty underlying the estimation of these parameters through a fully Bayesian approach, complemented with a sampling-based Markov Chain Monte Carlo (MCMC) method to address the lack of the analytical solution.

7.5.2 *Noise-Sensitive Sample Assignment*

One crucial piece of information that the basic grid-based decomposition process fails to account for is *potentially available knowledge* about the distribution of the noise across the input tensor. As discussed earlier, a sub-tensor which is poorly decomposed due to noise may negatively impact decomposition accuracies also for other parts of the tensor. Consequently, it is important to allocate resources to prevent a few noisy sub-tensors from negatively impacting the overall accuracy.

We note that there is a direct relationship between the amount of noise a sub-tensor has and the number of Gibbs samples it requires for accurate decomposition. In fact, the numbers of Gibbs samples allocated to different sub-tensors $\mathcal{X}_{\mathbf{k}}$ in

Algorithm 2 *do not need to be the same.* As we have seen in Sect. 7.5.1, Phase 1 decomposition of each sub-tensor is independent from the others and, thus, the number of Gibbs samples of different sub-tensors can be different. In fact, more samples can provide better accuracy for noisy sub-tensors and this can be used to improve the overall decomposition accuracy for a given number of Gibbs samples. Consequently, given a set of sub-tensors, with different amounts of noise, uniform assignment of the number of samples, $L = \left(\frac{L_{(\text{total})}}{|\mathcal{K}|}\right)$, where $L_{(\text{total})}$ is the total number of samples for the whole tensor and $|\mathcal{K}|$ is the number of sub-tensors, may not be the best choice. In this chapter, we rely on this key observation to help assign Gibbs samples to the various sub-tensors. On the other hand, the number of samples also directly impacts the cost of the probabilistic decomposition process. Therefore, the sample assignment process must be regulated carefully.

7.5.2.1 Naive Option: Noise Density-Based Sample Assignment

Intuitively, the number of samples a noisy sub-tensor, $\mathcal{X}_{\mathbf{k}}$, is allocated should be proportional to the density, $nd_{\mathbf{k}}$, of noise it contains:

$$L(\mathcal{X}_{\mathbf{k}}) = \lceil \gamma \times nd_{\mathbf{k}} \rceil + L_{\min}, \tag{7.6}$$

where $L_{\min}$ is the minimum number of samples a (non-noisy) tensor of the given size would need for accurate decomposition and γ is a control parameter. Note that the value of γ is selected such that the total number of samples needed is equal to the number, $L_{(\text{total})}$, of samples allocated for the whole tensor:

$$L_{(\text{total})} = \sum_{\mathbf{k} \in \mathcal{K}} L(\mathcal{X}_{\mathbf{k}}). \tag{7.7}$$

7.5.2.2 SIG-Based Sample Assignment: S-Strategy

Equations (7.6) and (7.7), above, help allocate samples across sub-tensors based on their noise densities. However, as discussed earlier, inaccuracies in decomposition of one sub-tensor can propagate across the rest of the sub-tensors in Phase 2. Therefore, a better approach would be to consider how errors can propagate across sub-tensors when allocating samples. More specifically, if we could assign a significance score to each sub-tensor, $\mathcal{X}_{\mathbf{k}}$, that takes into account not only its noise density, but also the position of the sub-tensor relative to other sub-tensors, we could use this information to better allocate the Gibbs samples to sub-tensors.

As discussed earlier in Sect. 7.3.3, the sub-tensor impact graph (SIG) of a given tensor can be used for assigning impact scores to each sub-tensor. This process, however, requires (in addition to the given SIG) a seed node set, $S \subseteq V$, which serves as the context in which scores are assigned: Given the SIG graph, $G(V, E)$,

and a set, $S \subseteq G(V, E)$, of seed nodes, the score $\mathbf{p}[i]$ of a node $v_i \in G(V, E)$ is obtained by solving $\mathbf{p} = (1 - \beta)\mathbf{A}\ \mathbf{p} + \beta\mathbf{s}$, where $\mathbf{A}$ denotes the transition matrix, β is a parameter controlling the overall importance of the seeds, and $\mathbf{s}$ is a seeding vector.

Our intuition is that we can use the sub-tensors with noise as the seeds in the above process. The naive way to create this seeding vector is to set $\mathbf{s}[i] = \frac{1}{\|S\|}$ if $v_i \in S$, and to $\mathbf{s}[i] = 0$, otherwise. However, we note that we can do better: given noise densities (nd) of the sub-tensors we can create a seeding vector

$$\mathbf{s}[\mathbf{k}] = \frac{nd_{\mathbf{k}}}{\sum_{\mathbf{j} \in \mathcal{K}} nd_{\mathbf{j}}},$$

and then, once the sub-tensor impact scores ($\mathbf{p}$) are computed, we can associate a noise-sensitive significance score,

$$\eta_{\mathbf{k}} = \frac{\mathbf{p}[\mathbf{k}] - min_{\mathbf{j} \in \mathcal{K}}(\mathbf{p}[\mathbf{j}])}{max_{\mathbf{j} \in \mathcal{K}}(\mathbf{p}[\mathbf{j}]) - min_{\mathbf{j} \in \mathcal{K}}(\mathbf{p}[\mathbf{j}])},$$

to each sub-tensor $\mathcal{X}_{\mathbf{k}}$. Given this score, we can then rewrite Eq. (7.6) as

$$L(\mathcal{X}_{\mathbf{k}}) = \lceil \gamma \times \eta_{\mathbf{k}} \rceil + L_{\min}. \tag{7.8}$$

7.5.3 Evaluation

In this section, we report experiments that aim to assess the proposed noise-sensitive sample assignment strategy (`s-strategy`) by comparing the performance of `nTD`, which leverages this strategy, against `GPTD` with uniform sample assignment and other naive strategies

7.5.3.1 Setup

Data Sets In these experiments, we used one user centered data set:*Ciao* [29]. The data is represented in the form of 5000 × 5000 × 996 (density 1.7×10^{-6}) tensor. Its schema is $\langle user, item, time \rangle$. In this data, the tensor cells contain rating values between 1 and 5 or (if the rating does not exist) a special "null" symbol.

Noise In these experiments, uniform value-independent type of noise was introduced by modifying the existing ratings in the data set. More specifically, given a uniform noise profile and density, we have selected a subset of the existing ratings (ignoring "null" values) and altered the existing values—by selecting a completely new rating (which we refer to as *value-independent noise*).

Evaluation Criteria We use the *root mean squares error* (RMSE) inaccuracy measure to assess the decomposition effectiveness. We also report the decomposition times. Unless otherwise reported, the execution time of the overall process is reported as if sub-tensor decompositions in Phase 1 and Phase 2 are all executed serially, without leveraging any sub-tensor parallelism. Each experiment was run ten times with different random noise distributions and averages are reported.

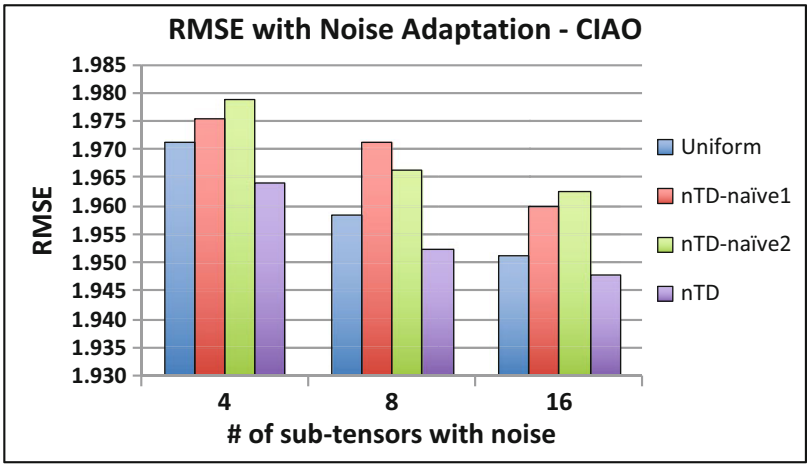
Hardware and Software We used a quad-core CPU Nehalem Node with 12.00GB RAM. All codes were run using Matlab R2015b. For conventional CP decomposition, we used MATLAB Tensor Toolbox Version 2.6 [1].

7.5.3.2 Discussion of the Results

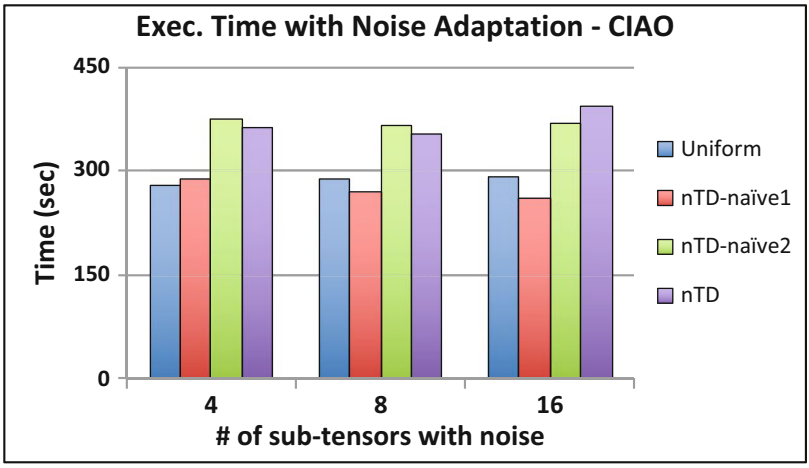
We start the discussion of the results by studying the impact of the `s-strategy` for leveraging noise profiles.

Impact of Leveraging Noise Profiles In Fig. 7.9, we compare the performance of `nTD` with noise-sensitive sample assignment (i.e., `s-strategy`) against `GPTD` with *uniform* sample assignment and the two naive noise adaptations, presented in Sects. 7.5.2.1 and 7.5.2.2, respectively. Note that in the scenario considered in this figure, we have 640 total Gibbs samples for 64 sub-tensors, providing on the average 10 samples per sub-tensor. In these experiments, we set $L_{\min}$ to 9 (i.e., very close to this average), thus requiring that $576 (= 64 \times 9)$ samples are uniformly distributed across the sub-tensors—this leaves only 64 samples to be distributed adaptively across the sub-tensors based on the noise profiles of the sub-tensors and their relationships to other sub-tensors. As we see in this figure, the proposed `nTD` is able to leverage these 64 uncommitted samples to significantly reduce RMSE relative to `GPTD` with *uniform* sample assignment. Moreover, we also see that naive noise adaptations can actually hurt the overall accuracy. `nTD`-naive uses biased sampling on the noise blocks and focuses on the centrality of sub-tensors. Thus, `nTD`-naive performs worse than uniform way. These together show that the proposed `s-strategy` is highly effective in leveraging rough knowledge about noise distributions to better allocate the Gibbs samples across the tensor.

In summary, the proposed sub-tensor impact graphs help allocate Gibbs samples in a way that takes into account how errors due to noise propagate across the whole tensor during the decomposition process.



(a)



(b)

Fig. 7.9 RMSE and execution time (without sub-tensor parallelism) for `nTD` with different num. of noisy sub-tensors ($4 \times 4 \times 4$ grid; uniform noise; value independent noise; noise density 10%; total num. of samples = 640; $L_{\min} = 9$, $F = 10$; max. num. of P2 iteration = 1000). (**a**) RMSE for Ciao. (**b**) Time for Ciao

7.6 Application #3: Personalized Tensor Decomposition (PTD) and Rank Assignment Based on Sub-Tensor Impact Scores

In many clustering applications, the user may have a focus of interest, i.e., part of the data for which the user needs high accuracy, and beyond this area focus, accuracy may not be as critical. Relying on this observation, in this section, we present a

personalized tensor decomposition (PTD) mechanism for accounting for the user's focus. Intuitively, the personalized tensor decomposition (PTD) algorithm partitions the tensor into multiple regions and then assigns different ranks to different sub-tensors: naturally, the higher the target rank is, the more accurate the decomposition of the sub-tensor. However, we note that preserving accuracy for foci of interest, while relaxing accuracy requirements for the rest of the input tensor is not a trivial task, especially because loss of accuracy at one region of the tensor may impact accuracies at other tensor regions. Therefore, PTD leverages sub-tensor impact graphs to account for the impact of the accuracy of one region of the tensor to the accuracies of the other regions of the tensor, each based on a different assumption about how the impact of inaccuracies propagates along the tensor. In particular, PTD analyzes the sub-tensor impact graph (in the light of the user's interest) to identify initial decomposition ranks for the sub-tensors in a way that will boost the final decomposition accuracies for partitions of interest.

7.6.1 Problem Formulation

Let an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ be partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\mathcal{X}_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$, where K_i indicates the number of partitions along mode-i, the size of the jth partition along mode i is $I_{j,i}$ (i.e., $\sum_{j=1}^{K_i} I_{j,i} = I_i$), and $\mathcal{K} = \{[k_{j_1}, \ldots, k_{j_i}, \ldots, k_{j_N}] \mid 1 \leq i \leq N,\ 1 \leq j_i \leq K_i\}$ is a set of sub-tensor indexes. The number, $\|\mathfrak{X}\|$, of partitions (and thus also the number, $\|\mathcal{K}\|$, of partition indexes) is $\prod_{i=1}^{N} K_i$. In addition, let $\mathcal{K}_P \subseteq \mathcal{K}$ be the set of sub-tensor indexes that indicate those sub-tensors for which the user requires higher accuracy. Note that, without loss of generality, we assume that the tensor $\mathcal{X}$ is re-ordered before it is partitioned in such a way that the number, K_i, of resulting partitions along each mode-i is minimal—i.e., along each mode, the entries of interest are clustered together to minimize the number of partitions.[6]

Given the above, let us use $\mathcal{X}_P$ as a shorthand to denote the cells of $\mathcal{X}$ collectively covered by the sub-tensors indexed by $\mathcal{K}_P$. The goal of the personalized tensor decomposition (PTD) is to obtain a *personalized (or preference sensitive) decomposition* $\hat{\mathcal{X}}$ of $\mathcal{X}$ in such a way that

$$accuracy(\mathcal{X}_P, \hat{\mathcal{X}}_P) > accuracy(\mathcal{X}_P, \tilde{\mathcal{X}}_P),$$

where $\hat{\mathcal{X}}_P$ is the reconstruction of the user selected region from the personalized decomposition $\hat{\mathcal{X}}$ and $\tilde{\mathcal{X}}_P$ is the reconstruction of the same region from a decomposition of $\mathcal{X}$ insensitive to the user preference $\mathcal{K}_P$ (or equivalently $\mathcal{K}_P = \mathcal{K}$). Naturally, we also aim that the time to obtain the personalized decomposition $\hat{\mathcal{X}}$ will be lesser than the time needed to obtain preference insensitive decomposition

[6]While this minimality criterion is not strictly required, the fewer partitions there are, the faster and potentially more effective will be the personalization process.

Algorithm 3 Overview of the PTD-CP process

Input: original tensor $\mathcal{X}$, partitioning pattern $\mathcal{K}$, user's focus $\mathcal{K}_P$, and decomposition rank, F
Output: personalized CP tensor decomposition $\hat{\mathcal{X}}$

1- compute the sub-tensor impact graph, $G(v, E, w())$, using the original tensor $\mathcal{X}$ and partitioning pattern $\mathcal{K}$;

2- for each partition $\mathbf{k} \in \mathcal{K}$, assign an initial decomposition rank $F_{\mathbf{k}}$ based on the sub-tensor impact graph, $G(v, E, w())$, and user's personalization focus $\mathcal{K}_P$;

3- obtain a block-based CP decomposition, $\hat{\mathcal{X}}$, of $\mathcal{X}$ using the partitioning pattern $\mathcal{K}$ and the initial decomposition ranks $\{F_{\mathbf{k}} \mid \mathbf{k} \in \mathcal{K}\}$;

return $\hat{\mathcal{X}}$

$\tilde{\mathcal{X}}$ and also that the personalized decomposition minimally impacts the rest of the tensor, i.e.,

$$accuracy(\mathcal{X}, \hat{\mathcal{X}}) \sim accuracy(\mathcal{X}, \tilde{\mathcal{X}}).$$

7.6.2 Sub-Tensor Rank Flexibility

Remember from Sect. 7.2.4, where we presented the update rules block-based tensor decomposition algorithms use for stitching the individual sub-tensor decompositions into a complete decomposition for the whole tensor, that (as visualized in Fig. 7.6) each $A^{(i)}_{(k_i)}$ is maintained incrementally by using, for all $1 \leq j \leq N$, the current estimates for $A^{(j)}_{(k_j)}$ and the decompositions in $\mathfrak{U}^{(j)}$, i.e., the F-rank sub-factors of the sub-tensors in $\mathfrak{X}$ along the different modes of the tensor. A closer look at the update rule for $A^{(i)}_{(k_i)}$ further reveals the following observation:

- *Sub-tensor Rank Flexibility:* One critical observation is that the above formulation *does not require that all sub-tensors in* $\mathfrak{X}$ *are decomposed with the same target rank* F.

  In fact, as long as one sub-tensor is decomposed to rank F, all other sub-tensors can be decomposed to ranks lesser than F and we can still obtain full F-rank factors, $A^{(i)}$, for $\mathcal{X}$.

7.6.3 Rank Assignment for Personalized Tensor Decomposition

The PTD algorithm relies on this sub-tensor rank flexibility property to personalize the block-based (CP) decomposition process described earlier: unlike the basic block-based scheme discussed in the introduction, PTD improves the accuracy

of the high-priority sub-tensors (indicated by $\mathcal{K}_P$) by assigning them higher initial decomposition ranks than the rest of the partitions (Algorithm 3). The **key difficulty**, however, is that one cannot arbitrarily reduce the decomposition ranks of low priority partitions, because the accuracy in one partition may impact final decomposition accuracies of other tensor partitions. As visualized in Algorithm 3, the proposed PTD algorithm first constructs a *sub-tensor impact graph*, G, that accounts for the propagation of inaccuracies along the tensor during a block-based decomposition process. The PTD algorithm then leverages this graph to account for the impact of the initial decomposition inaccuracy of one sub-tensor on the final decomposition accuracy of $\mathcal{X}_P$, i.e., the cells of $\mathcal{X}$ collectively covered by the user's declaration of interest (i.e., $\mathcal{K}_P$).

Intuitively, the initial decomposition rank, $F_{\mathbf{k}}$, of sub-tensor $\mathcal{X}_{\mathbf{k}}$ will need to reflect the impact of the initial decomposition of the sub-tensor $\mathcal{X}_{\mathbf{k}}$ on the final decomposition of the high-priority sub-tensors, $\mathcal{X}_{\boldsymbol{\kappa}}$, $\boldsymbol{\kappa} \in \mathcal{K}_P$. This implies that, when picking the decomposition ranks, $F_{\mathbf{k}}$, we need to measure how inaccuracies propagate within G over a large number of iterations of the alternating least squares (ALS) process. For this purpose we rely on the sub-tensor impact scores introduced in Sect. 7.3; more specifically, we compute the initial decomposition rank $F_{\mathbf{k}}$ of $\mathcal{X}_{\mathbf{k}}$ as

$$F_{\mathbf{k}} = \left\lceil F \times \frac{\mathbf{s}[k]}{\max_h \{\mathbf{s}[h]\}} \right\rceil,$$

where $\mathbf{s}[k]$ denotes the sub-tensor impact score of the sub-tensor $\mathcal{X}_{\mathbf{k}}$ in G. Intuitively, this formula sets the initial decomposition rank of the sub-tensor with the highest sub-tensor impact score (i.e., highest accuracy impact on the set of sub-tensors chosen by the user) to F, whereas other sub-tensors are assigned progressively smaller ranks (potentially all the way down to 1)[7] based on how far they are from the seed set in the sub-tensor impact graph, G.

7.6.4 Evaluation

In this section, we report sample results that aim to assess the effectiveness of the proposed personalized tensor decomposition (PTD) approach in helping preserve the tensor decomposition accuracy at parts of the tensor that are high-priority for the users.

[7] It is trivial to modify this equation such that the smallest rank will correspond to a user provided lower bound, $F_{\min}$, when such a lower bound is provided by the user.

Table 7.1 Various tensor partitioning scenarios considered in the experiments: the percentages are the sizes of the partitions (relative to the overall size of the mode) along each mode

$2 \times 2 \times 2$ partitions		
Configuration	Part. #1	Part. #2
1 - Most balanced	50%	50%
2 -	40%	60%
3 -	30%	70%
4 -	25%	75%
5 -	10%	90%
6 - Least balanced	1%	99%

7.6.4.1 Setup

Data Set In the experiments reported in this chapter, we used the *Ciao* [29] data set, represented in the form of a $167 \times 967 \times 18$ (density 2.2×10^{-4}) tensor, with the schema $\langle user, item, category \rangle$. In these experiments, we assume that the input tensor is partitioned into 8 ($= 2 \times 2 \times 2$) according to the scenarios shown in Table 7.1 and two randomly selected sub-tensors are marked as more important than the rest.

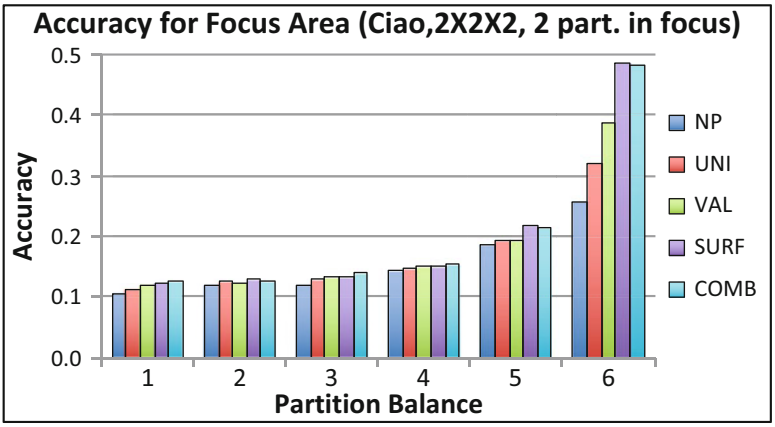
Decomposition Strategies We considered five decomposition strategies: not personalized (`NP`), uniform edge weights (`UNI`), surface of interaction-based edge weights (`SURF`), value alignment-based edge weights (`VAL`), and combined edge weights (`COMB`). The target CP decomposition rank, F, is set to 10.

Evaluation Criteria We use the measure reported in Sect. 7.2.2.2 to assess decomposition accuracy and execution time. In particular, we report accuracies for both user's area of focus and the whole tensor.

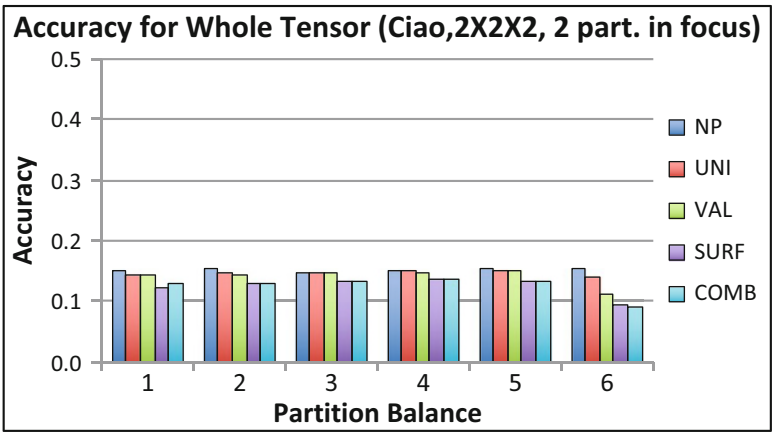
Hardware and Software We ran the experiments reported in this section on a quad-core Intel(R) Core(TM)i5-2400 CPU @ 3.10GHz machine with 8.00GB RAM. All codes were implemented in Matlab and run using Matlab R2015b. For CP decomposition, we used MATLAB Tensor Toolbox Version 2.6 [1].

7.6.4.2 Discussion of the Results

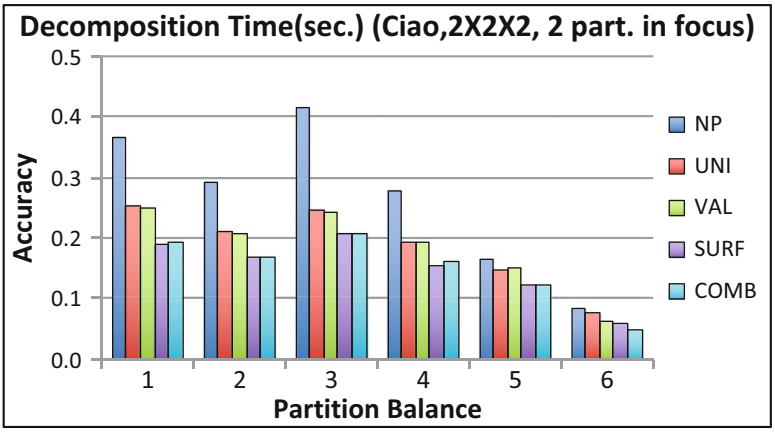
Sub-tensor impacts help take into account how inaccuracies in the decomposition propagate into high- and-low-priority regions. We therefore expect that allocating resources using sub-tensor impact graphs should provide better accuracies for high-priority regions. As we see in Fig. 7.10, as expected, PTD algorithms boost accuracy for the high-priority partitions in the user focus, especially where the partitions are of heterogeneous sizes (as is likely to be the case in real situations). While, as would be expected, the PTD algorithms have impact on the overall decomposition accuracy for the whole tensor, this is more than compensated by gains in accuracies in high-priority areas. Moreover, the figure also shows that the gains in accuracy in high-priority partitions within the user's focus come also with significant gains in execution times for the decomposition process.



(a)



(b)



(c)

Fig. 7.10 Experiment results with two partitions in focus. (**a**) Accuracy of the region of focus. (**b**) Whole tensor accuracy. (**c**) Decomposition time

The figure also establishes that, both in terms of accuracy and execution time gains, we can order the various edge weighting strategies as follows: `UNI` (least effective), `VAL`, `SURF`, and `COMB` (most effective). In other words, as we argued in Sect. 7.3.2.4, the most effective way to account for the propagation of inaccuracies is to combine the surface of interaction and value alignment-based edge weights into a single weight which accounts for both shapes of the sub-tensors and their value alignments.

7.7 Conclusions

Computational complexity of tensor decomposition is a major bottleneck in many applications. Block-based tensor decomposition is employed to efficiently conduct tensor decomposition for large-scale data analysis. However, we need a smart strategy to account for the relationship among these sub-tensors(blocks). Therefore, we proposed sub-tensor impact graph (SIG) to account for the propagation of impacts within the sub-tensors during the decomposition process. Then, we presented three applications of SIG to efficiently solve the challenges in personalized tensor analysis, incremental tensor analysis, and noise tensor analysis. Experiments results on real data sets show that SIGs can improve the performance of tensor analysis in these three applications in both of execution time and decomposition accuracy.

Finally, we would like to note that, here we presented three distinct uses of sub-tensor impact graphs for three distinct challenges (dynamic data, noisy data, and personalization). In practice, there is no reason these approaches cannot be combined to tackle more complex scenarios, such as personalized clustering and analysis over dynamically evolving data sets. We leave the study of these more complex scenarios as future work.

Acknowledgements Research is supported by NSF#1318788 "Data Management for Real-Time Data Driven Epidemic Spread Simulations," NSF#1339835 "E-SDMS: Energy Simulation Data Management System Software," NSF#1610282 "DataStorm: A Data Enabled System for End-to-End Disaster Planning and Response," NSF#1633381 "BIGDATA: Discovering Context-Sensitive Impact in Complex Systems," and "FourCmodeling": EUH2020 Marie Sklodowska-Curie grant agreement No 690817.

References

1. B.W. Bader, T.G. Kolda et al., MATLAB Tensor Toolbox Version 2.5. Available online (January 2012)
2. A. Balmin, V. Hristidis, Y. Papakonstantinou. ObjectRank: authority-based keyword search in databases, in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)* (2004)

3. X. Cao, X. Wei, Y. Han, D. Lin, Robust face clustering via tensor decomposition. IEEE Trans. Cybern. **45**(11), 2546–2557 (2015)
4. S. Chakrabarti, Dynamic personalized pagerank in entity-relation graphs, in *Proceeding WWW '07 Proceedings of the 16th International Conference on World Wide Web* (2007)
5. X. Chen, K.S. Candan, LWI-SVD: low-rank, windowed, incremental singular value decompositions on time-evolving data sets, in *KDD '14 Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014)
6. I. Davidson, S. Gilpin, O. Carmichael, P. Walker, Network discovery via constrained tensor analysis of FMRI data, in *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 194–202 (2013)
7. C. Ding, X. He, K-means clustering via principal component analysis, in *ICML '04 Proceedings of the Twenty-First International Conference on Machine Learning* (2004)
8. P. Drineas, A. Frieze, R. Kannan, S. Vempala, V. Vinay, Clustering large graphs via the singular value decomposition. Mach. Learn. **56**, 9–33 (2004)
9. F.M. Harper, J.A. Konstan, The MovieLens datasets: history and context. Trans. Interact. Intell. Syst. **5**, 19:1–19:19 (2015)
10. R.A. Harshman, Foundations of the PARAFAC procedure: model and conditions for an explanatory multi-mode factor analysis. *UCLA Working Papers in Phonetics*, vol. 16 (1970), pp. 1–84
11. S. Huang, K.S. Candan, M.L. Sapino, BICP: block-incremental CP decomposition with update sensitive refinement, in *Proceeding CIKM '16 Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (2016)
12. I. Jeon, E. Papalexakis, U. Kang, C. Faloutsos, HaTen2: billionscale tensor decompositions, in *Proceedings - International Conference on Data Engineering (ICDE)* (2015)
13. B. Jeon, I. Jeon, L. Sael, U. Kang, SCouT: scalable coupled matrix-tensor factorization - algorithm and discoveries, in *IEEE 32nd International Conference on Data Engineering (ICDE)* (2016)
14. U. Kang, E.E. Papalexakis, A. Harpale, C. Faloutsos, Gigatensor: scaling tensor analysis up by 100 times algorithms and discoveries, in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 316–324 (2012)
15. M. Kim, K.S. Candan, Decomposition by normalization (DBN): leveraging approximate functional dependencies for efficient CP and tucker decompositions. Data Min. Knowl. Disc. **30**(1), 1–46 (2016)
16. T.G. Kolda, B.W. Bader, Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
17. T.G. Kolda, J. Sun, Scalable tensor decompositions for multi-aspect data mining, in *Eighth IEEE International Conference on Data Mining (ICDM)* (2008)
18. X. Li, S.Y. Huang, K.S. Candan, M.L. Sapino, Focusing decomposition accuracy by personalizing tensor decomposition (PTD), in *Proceeding CIKM '14 Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management* (2014)
19. X. Li, K.S. Candan, M.L. Sapino, nTD: noise-profile adaptive tensor decomposition, in *Proceeding WWW '17 Proceedings of the 26th International Conference on World Wide Web* (2017)
20. S. Papadimitriou, J. Sun, C. Faloutsos, Streaming pattern discovery in multiple time-series, in *Proceeding VLDB '05 Proceedings of the 31st International Conference on Very Large Data Bases* (2015)
21. E. Papalexakis, C. Faloutsos, N. Sidiropoulos, Parcube: sparse parallelizable tensor decompositions, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pp. 521–536 (2012)
22. I. Perros, E.E. Papalexakis, F. Wang, R. Vuduc, E. Searles, M. Thompson, J. Sun, Spartan: scalable parafac2 for large & sparse data (2017). arXiv preprint arXiv:1703.04219
23. A.H. Phan, A. Cichocki, PARAFAC algorithms for large-scale problems. Neurocomputing **74**(11), 1970–1984 (2011)
24. C.E. Priebe et al., Enron data set (2006). http://cis.jhu.edu/parky/Enron/enron.html

25. R. Salakhutdinov, A. Mnih, Probabilistic matrix factorization, in *Proceeding NIPS'07 Proceedings of the 20th International Conference on Neural Information Processing Systems* (2007)
26. J. Sun, S. Papadimitriou, P.S. Yu, Window based tensor analysis on high dimensional and multi aspect streams, in *Sixth International Conference on Data Mining (ICDM'06)*, pp. 1076–1080 (2006)
27. J. Sun, D. Tao, S. Papadimitriou, P.S. Yu, C. Faloutsos, Incremental tensor analysis: theory and applications. ACM Trans. Knowl. Discov. Data **2**(3), Article No. 11 (2008)
28. Y. Sun, J. Gao, X. Hong, B. Mishra, B. Yin, Heterogeneous tensor decomposition for clustering via manifold optimization. IEEE Trans. Pattern Anal. Mach. Intell. **38**(3), 476–489 (2016)
29. J. Tang et al., Trust & distrust computing dataset (2011). https://www.cse.msu.edu/~tangjili/trust.html
30. C.E. Tsourakakis, Mach: fast randomized tensor decompositions (2009). Arxiv preprint arXiv:0909.4969
31. L. Tucker, Some mathematical notes on three-mode factor analysis. Psychometrika **31**, 279–311 (1966)
32. J. Wu, Z. Wang, Y. Wu, L. Liu, S. Deng, H. Huang, Tensor CP decomposition method for clustering heterogeneous information networks via stochastic gradient descent algorithms. Sci. Program. **2017**, 13 (2017), Article ID 2803091
33. L. Xiong et al., Temporal collaborative filtering with Bayesian probabilistic tensor factorization, in *Proceedings of the 2010 SIAM International Conference on Data Mining* (2010)