BICP: Block-incremental CP decomposition with update sensitive refinement

BICP: Block-Incremental CP Decomposition with Update Sensitive Refinement

ABSTRACT

With many applications relying on multi-dimensional datasets for decision making, tensors (or multi-dimensional arrays) are emerging as a popular data representation to support diverse types of data, such as sensor streams and social networks. Consequently, tensor decomposition forms the basis for many data analysis and knowledge discovery tasks, from clustering, trend detection, anomaly detection, to correlation analysis. In applications where data evolves over time and the tensor-based analysis results need to be continuously maintained, re-computation of the whole tensor decomposition with each update will cause high computational costs and incur large memory overheads. In this paper, we propose a two-phase block-incremental CP-based tensor decomposition technique, BICP, that efficiently and effectively maintains tensor decomposition results in the presence of dynamically evolving tensor data. In its first phase, instead of repeatedly conducting ALS on each sub-tensor, BICP only revises the decompositions of the tensors that contain updated data. Moreover, when updates are relatively small with respect to the block size, BICP relies on a incremental factor tracking to avoid re-decomposition the updated sub-tensor. In its second phase, BICP limits the block-centric refinement process to only those blocks that are critical given the update. Experiment results show that the proposed method significantly reduces the execution time while assuring high accuracy.

1. INTRODUCTION

With many applications relying on multi-dimensional datasets for decision making, tensors (or multi-dimensional arrays) are emerging as a popular data representation [35, 17, 14, 16, 19]. Matrix-shaped data (i.e., 2-mode tensors) are often analyzed for their latent semantics through matrix decomposition operations, such as singular value decomposition (SVD). The corresponding analysis operation which applies to tensors with more than two modes is known as *tensor decomposition*, such as CP (Figure 1) and Tucker decompositions [36]. These form the basis for many data analysis and knowledge discovery tasks, from clustering, trend and anomaly detection [17] to correlation analysis [30].

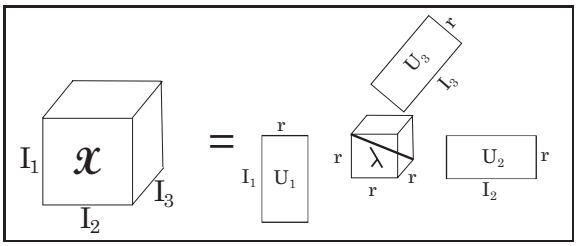


Figure 1: CP-decomposition of a 3-mode tensor [7, 12] results in a diagonal core and three factor matrices

A critical challenge for tensor based analysis is its computational complexity and decomposition can be a bottleneck in some applications. Especially when data evolves over time and the tensor-based analysis results need to be continuously maintained, re-decomposition of the whole tensor with each and every update will incur high computational costs. In this paper, we propose a two-phase block-incremental tensor decomposition technique that efficiently and effectively maintains tensor decomposition results in the presence of dynamically evolving tensor data.

1.1 Key Observations

[20] and [25] presented a block-based alternative to tensor decomposition: (a) in their first phase, these partition the input tensor into pieces and obtain (potentially in parallel) decompositions for each piece; (b) in the second phase, they stitch the partial decomposition results into a combined decomposition through an iterative block-centric refinement process. In this paper, we argue that, when extended with methods to eliminate waste and support reuse, block-based tensor decomposition can provide an effective framework for incremental tensor analysis. We use the example in Figure 2 to illustrate this:

- Let us assume that all the updates on the tensor are limited to the blue block: since in Phase 1, each block is decomposed separately, in this situation, there would not be a need to recompute other blocks' decompositions.
- In fact, if the update on the blue block is relatively small, we may be able to completely avoid re-decomposing this block and, instead, we can maintain the sub-tensor incrementally.

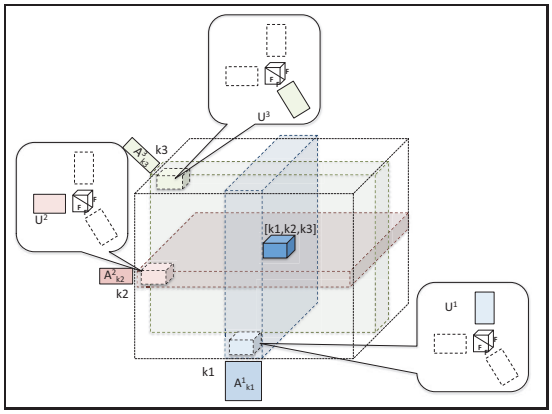


Figure 2: [20, 25] divide a given tensor into blocks/sub-tensors; decompose each of these sub-tensors separately, and then stitch these together through an iterative refinement process, where a given sub-tensor decomposition is refined by leveraging decompositions of other sub-tensors aligned with the block along different modes

- Furthermore, in Phase 2 of the process, we may be able to limit the refinement process only to those blocks that are aligned with the updated block along the different modes to save significant time, while preserving accuracy (in fact, as we will see in this paper, a subset of those blocks may often be sufficient).

1.2 Contributions: Two-Phase Block-Incremental Tensor Decomposition

Let us assume that we are given a tensor, $\mathcal{X}$, with decomposition, $\hat{\mathcal{X}}$, and an update, Δ, on the tensor. $\hat{\mathcal{X}}$. Based on the above observations, in this paper, we present a two-phase block-based incremental CP tensor decomposition (BICP) approach which significantly reduces computational cost of obtaining the decomposition of the updated tensor, while maintaining high accuracy:

- **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 tensors that contain updated data. Moreover, when updates are relatively small with respect to the block size, BICP relies on an incremental factor tracking to avoid re-decomposition of the updated sub-tensor.

- **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 propose a measure of "impact likelihood" and use this 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 likelihood score: naturally, the larger the impact likelihood of a sub-tensor, 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 decomposition process is fast and accurate.

In the next section, we present the related work. In Section 3, we present the relevant background, notations and formalize the problem. In Section 4, we introduce the proposed two-phase block-incremental tensor decomposition technique. Experiment results, reported in Section 5, show that the proposed algorithms significantly reduce the amount of execution time while assuring the accuracy. We conclude the paper in Section 6.

2. RELATED WORK

Tensor based representations of data and tensor decompositions (especially the two widely used decompositions CP [12] and Tucker [36]) are proven to be effective in multi-aspect data analysis for capturing high-order structures in multi-dimensional data [16, 34]. For example, in [28], authors analyze an email social network using tensor decomposition. In [1], authors introduce a tensor-based framework to identify epileptic seizures and, in [34], authors use tensors to incorporate user click information to improve web search. [11] shows that decomposition of fMRI data can reproduce well known results in neurology for differentiating healthy and Alzheimer affected individuals.

There are two widely used toolboxes: the *Tensor Toolbox for Matlab* [3] (for sparse tensors) and *N-way Toolbox for Matlab* [2] (for dense tensors). Since tensor decomposition is a costly process for both sparse and dense tensors, various optimization and parallel algorithms and systems have been developed. [17] proposed a memory-efficient Tucker (MET) decomposition to address the intermediate blowup problem in Tucker decomposition by updating a subset of the modes at a time. [35] proposed MACH, a randomized algorithm (based on randomized sampling) that speedups the Tucker decomposition while providing accuracy guarantees. Recently, [24] proposed a fast approach for CP that decomposes an unfolded tensor in lower order, instead of directly factorizing the high order tensor. TensorDB [15] extends a block-based array store to store and retrieve data and introduces optimization schemes for efficient CP-ALS based in-database tensor decompositions. [19] proposed a Personalized Tensor Decomposition (PTD) mechanism that boosts accuracy and reduces execution time in situations where the user's interest is not uniformly distributed across the whole tensor. Parallelization of tensor decompositions have been proposed for different platforms [4, 10, 33]. In [26, 31], authors propose PARCUBE, a sampling based, parallel and sparsity promoting, approximate PARAFAC decomposition scheme. [13] proposed HaTen2, a massively distributed MapReduce based implementation of PARAFAC and Tucker running on the MapReduce platform.

Due to the intrinsic computational complexity of tensor decomposition, efficient incremental tensor decomposition is necessary in many applications such as video tracking, foreground detection, and face recognition [18, 21, 22, 29].

[23] presented a pioneering work on updating a tensor with PARAFAC decomposition, and applied it to MIMO radar application. [32] proposed tensor update algorithms: Dynamic Tensor Analysis (DTA), Streaming Tensor Analysis (STA), and Window-based Tensor Analysis (WTA) where DTA obtains an update factor matrix by extracting leading eigenvectors of incrementally maintained covariance matrix in each mode. STA is a fast algorithm of an approximate DTA by SPIRIT algorithm and WTA combines the idea of sliding windows with DTA. [22] considered the third-order tensor update and downdate problems when a new arrival of the data forms a matrix.

3. BACKGROUND AND NOTATIONS

We now present the relevant background and notations.

3.1 Tensors and Tensor Decompositions

Tensors are generalizations of matrices: while a matrix is essentially a 2-mode array, a tensor is an array of larger number of modes. Intuitively, the tensor model maps a schema with N attributes to an N-modal array (where each potential tuple is a tensor cell).

The two most popular tensor decomposition algorithms are the Tucker [36] and the CANDECOMP/PARAFAC (CP) [12] decompositions. Intuitively, both generalize singular value matrix decomposition (SVD) to tensors.

3.2 CP Decomposition

As shown in Figure 1, given a tensor $\boldsymbol{\mathcal{X}}$, 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 $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. CP would decompose $\boldsymbol{\mathcal{X}}$ into $\mathring{\boldsymbol{\mathcal{X}}}$ consisting of three matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$, such that

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

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

The alternating least squares (ALS) method is often used for obtaining tensor decomposition: at each iteration, ALS estimates one factor matrix while maintaining other matrices fixed; this process is repeated for each factor matrix associated to the modes of the input tensor until convergence condition is reached. Since tensor decomposition is an approximation algorithm, the new tensor $\tilde{\boldsymbol{\mathcal{X}}}$ obtained by recomposing the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ is often different from the input tensor, $\boldsymbol{\mathcal{X}}$. The accuracy of the decomposition is often measured by considering the Frobenius norm of the difference tensor:

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

3.3 Block-based CP Decomposition

As discussed previously, block-based CP 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. Let us consider an N-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, partitioned into a set (or grid) of sub-tensors

Algorithm 1 Two-Phase Block-based CP Decomp. [20]

Input: original tensor, $\boldsymbol{\mathcal{X}}$; partitioning pattern, $\mathcal{K}$; and decomposition rank, F

Output: CP tensor decomposition $\mathring{\boldsymbol{\mathcal{X}}}$

1. *Phase 1:* for all $\vec{k} \in \mathcal{K}$
   - decompose $\boldsymbol{\mathcal{X}}_{\vec{k}}$ into $\boldsymbol{U}^{(1)}_{\vec{k}}$, $\boldsymbol{U}^{(2)}_{\vec{k}}$, …, $\boldsymbol{U}^{(N)}_{\vec{k}}$
2. *Phase 2:* repeat for each $\vec{k} = [k_1, \ldots, k_N] \in \mathcal{K}$
   (a) for each mode $i = 1$ to N
      i. refine $\boldsymbol{A}^{(i)}_{(k_i)}$ using $\boldsymbol{U}^{(i)}_{[*,\ldots,*,k_i,*,\ldots,*]}$, for each block $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$; more specifically,
         - compute $\boldsymbol{T}^{(i)}_{(k_i)}$, which involves the use of $\boldsymbol{U}^{(i)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ (i.e. the mode-i factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$)
         - revise $\boldsymbol{P}_{[*,\ldots,*,k_i,*,\ldots,*]}$ and $\boldsymbol{Q}_{[*,\ldots,*,k_i,*,\ldots,*]}$ using $\boldsymbol{U}^{(i)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ and $\boldsymbol{A}^{(i)}_{(k_i)}$
         - compute $\boldsymbol{S}^{(i)}_{(k_i)}$ using the above
         - refine $\boldsymbol{A}^{(i)}_{(k_i)}$ using the above
      ii. for all $\vec{l} = [*, \ldots, *, k_i, *, \ldots, *] \in \mathcal{K}$
         - refine $\boldsymbol{P}_{\vec{l}}$ and $\boldsymbol{Q}_{\vec{l}}$ using
           – $\boldsymbol{U}^{(i)}_{\vec{l}}$ and $\boldsymbol{A}^{(i)}_{(k_i)}$

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

$\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ 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 $\boldsymbol{\mathcal{X}}$. Let us further assume that each sub-tensor in $\mathfrak{X}$ has already been decomposed with target rank F and let $\mathfrak{U}^{(i)} = \{\boldsymbol{U}^{(i)}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ 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 $\boldsymbol{\mathcal{X}}_{\vec{k}}$, we have

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

where $\boldsymbol{I}$ 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, [25] presents an iterative improvement algorithm for composing these initial sub-factors into the full F-rank factors, $\boldsymbol{A}^{(i)}$ (each one along one mode), for the input tensor, $\boldsymbol{\mathcal{X}}$. The outline of this block based process is as follows: Let us partition each factor $\boldsymbol{A}^{(i)}$ into K_i parts corresponding to the block boundaries along mode i:

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

Given this partitioning, each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, $\vec{k} = [k_1, \ldots, k_i, \ldots, k_N] \in \mathcal{K}$ can be described in terms of these sub-factors (Figure 2):

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

Moreover, the current estimate of the sub-factor $\boldsymbol{A}^{(i)}_{(k_i)}$ can be revised using the following refinement rule:

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

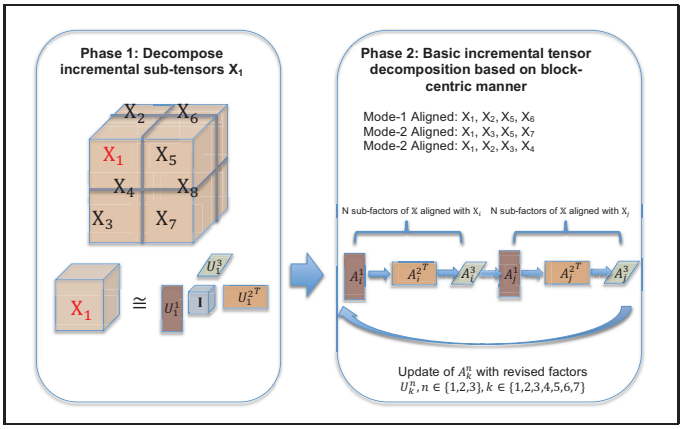


Figure 3: Illustration of basic method of incremental block-based tensor decomposition(the notation is introduced in Section 3.2)

$$\begin{aligned}
\boldsymbol{A}^{(i)}_{(k_i)} &\longleftarrow \boldsymbol{T}^{(i)}_{(k_i)} \left(\boldsymbol{S}^{(i)}_{(k_i)}\right)^{-1}, \quad \text{where} \\
\boldsymbol{T}^{(i)}_{(k_i)} &= \sum_{\vec{l}\in\{[*,\ldots,*,k_i,*,\ldots,*]\}} \boldsymbol{U}^{(i)}_{\vec{l}} \left(\boldsymbol{P}_{\vec{l}} \oslash (\boldsymbol{U}^{(i)T}_{\vec{l}} \boldsymbol{A}^{(i)}_{(k_i)})\right) \\
\boldsymbol{S}^{(i)}_{(k_i)} &= \sum_{\vec{l}\in\{[*,\ldots,*,k_i,*,\ldots,*]\}} \boldsymbol{Q}_{\vec{l}} \oslash \left(\boldsymbol{A}^{(i)T}_{(k_i)} \boldsymbol{A}^{(i)}_{(k_i)}\right).
\end{aligned}$$

Above, given $\vec{l} = [l_1, l_2, \ldots, l_N]$, we have $\boldsymbol{P}_{\vec{l}} = \circledast_{h=1}^{N}(\boldsymbol{U}^{(h)T}_{\vec{l}} \boldsymbol{A}^{(h)}_{(l_h)})$ and $\boldsymbol{Q}_{\vec{l}} = \circledast_{h=1}^{N}(\boldsymbol{A}^{(h)T}_{(l_h)} \boldsymbol{A}^{(h)}_{(l_h)})$. Here, $\circledast$ denotes the Hadamart product and $\oslash$ denotes the element-wise division operation. [20] leverages this refinement rule to develop a two-phase block-centric decomposition algorithm (Algorithm 1). While the precise derivation of the above refinement rules and the details of the algorithm are not critical for our discussion (and is beyond the scope of this paper), as we see in the next section, this two-phase framework enables us to develop an efficient block-based incremental tensor decomposition process.

4. BLOCK-BASED INCREMENTAL CP (BICP) DECOMPOSITION

Let us assume that we are given a tensor, $\boldsymbol{\mathcal{X}}$, with decomposition, $\mathring{\boldsymbol{\mathcal{X}}}$, and an update, Δ, on the tensor. In this section, we propose a two-phase block-incremental CP tensor decomposition (BICP) approach to obtain the decomposition of the updated tensor with high efficiency, while maintaining high accuracy. Below, we formalize the key observations underlying the proposed method:

• **Observation #1 (Update Sensitive Decompositions in Phase 1):** As described in the previous section, in Phase 1 each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, where $\vec{k} \in \mathcal{K}$ such that $\mathcal{K}$ is a partitioning pattern, needs to be decomposed into $\boldsymbol{U}^{(1)}_{\vec{k}}$, $\boldsymbol{U}^{(2)}_{\vec{k}}$, …, $\boldsymbol{U}^{(N)}_{\vec{k}}$. The fact that each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$ can be decomposed independently from the others means that any updates on the values of one sub-tensor will have no effect on other sub-tensors' Phase 1 decompositions. Therefore, to improve efficiency, in this phase only sub-tensors which have been modified need to be re-decomposed.

This is visualized in Figure 3: In this example, the given tensor $\boldsymbol{\mathcal{X}}$ is partitioned into eight sub-tensors, $\boldsymbol{\mathcal{X}}_1$ through $\boldsymbol{\mathcal{X}}_8$. For this example, let us assume that all the updates are contained in sub-tensor $\boldsymbol{\mathcal{X}}_1$. Instead of conducting CP decomposition on all 8 sub-tensors, only sub-tensor $\boldsymbol{\mathcal{X}}_1$ needs to be decomposed to recompute sub-factors $\boldsymbol{U}^{(1)}_{(1)}$, $\boldsymbol{U}^{(2)}_{(1)}$, $\boldsymbol{U}^{(3)}_{(1)}$ of sub-tensor $\boldsymbol{\mathcal{X}}_1$. Sub-factors of all other sub-tensors can be inherited from the original tensor decomposition since there are no updates in other sub-tensors. In fact, as we later see in Section 4.1, under certain conditions, we can further save processing, by revising the decomposition of $\boldsymbol{\mathcal{X}}_1$ through a tracking algorithm rather than executing a full re-decomposition of the sub-tensor.

Algorithm 2 The outline of the basic block-centric incremental tensor decomposition algorithm

Input: original tensor, $\boldsymbol{\mathcal{X}}$; tensor block partitioning pattern, $\mathcal{K}$; rank, F, block decomposition, $\mathring{\boldsymbol{\mathcal{X}}} = \langle U, P, Q, A\rangle$, of $\boldsymbol{\mathcal{X}}$; and a tensor update, Δ.

Output: CP tensor decomposition $\mathring{\boldsymbol{\mathcal{X}}}_{\Delta}$ of the updated tensor

1. Let $\mathcal{T}$ be the set of sub-tensors containing the update Δ
2. *Phase 1:* for all $\vec{t} \in \mathcal{T}$
   - decompose $\boldsymbol{\mathcal{X}}_{\vec{t}}$ into $\boldsymbol{U}^{(1)}_{\vec{t}}$, $\boldsymbol{U}^{(2)}_{\vec{t}}$, …, $\boldsymbol{U}^{(N)}_{\vec{t}}$
3. *Phase 2:*
   (a) $\mathcal{R} = \bigcup_{\vec{t}\in\mathcal{T}}$ *direct_impact*$(\boldsymbol{\mathcal{X}}_{\vec{t}})$
   (b) Repeat for $\vec{r} \in \mathcal{R}$
      i. for each mode $1 \le m \le N$ for which $\vec{r}$ is aligned with any $\vec{t} \in \mathcal{T}$
         A. update $\boldsymbol{A}^{(m)}_{(r_m)}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$, for each block $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,r_m,*,\ldots,*]} \in \mathcal{R}$; more specifically,
            - compute $\boldsymbol{T}^{(m)}_{(r_m)}$, which involves the use of $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$ (i.e. the mode-i factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,r_m,*,\ldots,*]}$)
            - revise $\boldsymbol{P}_{[*,\ldots,*,r_m,*,\ldots,*]}$ and $\boldsymbol{Q}_{[*,\ldots,*,r_m,*,\ldots,*]}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$ and $\boldsymbol{A}^{(m)}_{(r_m)}$
            - compute $\boldsymbol{S}^{(m)}_{(r_m)}$ using the above
            - update $\boldsymbol{A}^{(m)}_{(r_m)}$ using the above
         B. for all $\vec{l} = [*,\ldots,*,r_i,*,\ldots,*] \in \mathcal{R}$
            - update $\boldsymbol{P}_{\vec{l}}$ and $\boldsymbol{Q}_{\vec{l}}$ using
               – $\boldsymbol{U}^{(m)}_{\vec{l}}$ and $\boldsymbol{A}^{(m)}_{(r_m)}$

   until stopping condition
4. Return $\mathring{\boldsymbol{\mathcal{X}}}_{\Delta}$

• **Observation #2 (Update Sensitive Refinement in Phase 2):** A close look at Phase 2 of Algorithm 1 shows that, the refinement of a block $\boldsymbol{\mathcal{X}}_{\vec{k}}$ involves the refinements of its related factors $\boldsymbol{A}^{(i)}_{(k_i)}$ for each mode $i = 1$ to N:

$$\boldsymbol{\mathcal{X}}_{\vec{k}} \approx \boldsymbol{A}^{(1)}_{(k_1)} \times_2 \boldsymbol{A}^{(2)}_{(k_2)} \cdots \times_N \boldsymbol{A}^{(N)}_{(k_N)}.$$

$\boldsymbol{A}^{(i)}_{(k_i)}$, in turn, depends on the sub-factors of the sub-tensors contributing to its refinement . Given a sub-tensor, $\boldsymbol{\mathcal{X}}_{\vec{k}}$, we say that those sub-tensors that are aligned with **any** of the modes of $\boldsymbol{\mathcal{X}}_{\vec{k}}$ have *direct impact* on $\boldsymbol{\mathcal{X}}_{\vec{k}}$:

$$\textit{direct\_impact}(\boldsymbol{\mathcal{X}}_{\vec{k}}) = \left\{ \boldsymbol{\mathcal{X}}_{\vec{j}} (\neq \boldsymbol{\mathcal{X}}_{\vec{k}}) \middle| \exists_{1\le i\le N} \;\; k_i = j_i \right\}.$$

This is visualized in Figures 2 and 3. Let $\boldsymbol{\mathcal{X}}_1$ be the modified sub-tensor in Figure 3: In this example, $\boldsymbol{\mathcal{X}}_2$, $\boldsymbol{\mathcal{X}}_5$ and $\boldsymbol{\mathcal{X}}_6$ are aligned with $\boldsymbol{\mathcal{X}}_1$ on mode-1; $\boldsymbol{\mathcal{X}}_3$, $\boldsymbol{\mathcal{X}}_5$, and $\boldsymbol{\mathcal{X}}_7$ are

Algorithm 3 Incremental factor tracking

Input: the update on the sub-tensor, Δ_t; existing sub-tensor decomposition factors $\boldsymbol{U}_1$ through $\boldsymbol{U}_N$; corresponding energy matrices $\mathcal{S}_1$ through $\mathcal{S}_N$; decomposition rank F; and forgetting factor λ

Output: updated sub-tensor decomposition factors $\boldsymbol{U}'_1$ through $\boldsymbol{U}'_N$; revised energy matrices $\mathcal{S}'_1$ through $\mathcal{S}'_N$

1. Let $\mathcal{C}$ be update-critical fibers of Δ_t, with highest energy
2. for all modes $m = 1$ to N
   (a) $\Delta_m = matricize(\Delta_t, m)$
   (b) Obtain energy matrix $\mathcal{S}_m$ for $\boldsymbol{U}_m$ {compute or copy from previous time step}
   (c) for all columns $j = 1$ to I_m
   - if the column j of the matricization Δ_m corresponds to a fiber in $\mathcal{C}$
     i. Initialize the update vector $x_1 := \Delta_m[j]$
     ii. for each basis vector $i = 1$ to rank F
     - $y_i := \boldsymbol{U}_m[i]^T x_i$ {project onto basis vector}
     - $\mathcal{S}'_m[i] \leftarrow \lambda \mathcal{S}[i] + y_i^2$ {revise energy}
     - $e_i := x_i - y_i \boldsymbol{U}_m[i]$ {compute error}
     - $\boldsymbol{U}'_m[i] \leftarrow \boldsymbol{U}_m[i] + \frac{1}{\mathcal{S}'_m[i]} y_i e_i$ {revise basis vector}
     - $x_{i+1} := x_i - y_i \boldsymbol{U}'_m[i]$ {revise the update vector to reflect the revised basis vector}

aligned with $\boldsymbol{\mathcal{X}}_1$ on mode-2; $\boldsymbol{\mathcal{X}}_2$, $\boldsymbol{\mathcal{X}}_3$, and $\boldsymbol{\mathcal{X}}_4$ are aligned with $\boldsymbol{\mathcal{X}}_1$ on mode-3. We argue that identifying such direct relationships among sub-tensors and leveraging these to manipulate the refinement process can help significantly reduce the redundant work, while maintaining high accuracy.

• **Outline of Basic BICP:** The outline of the basic block based incremental tensor decomposition algorithm is presented in Algorithm 2 and visualized in Figure 3. This basic outline, however, does not include several key optimizations on Phase 1 and Phase 2, which, as we experimentally validate in Section 5, provide significant savings. We discuss these optimizations in the rest of the paper.

4.1 Optimization #1 - Incremental Factor Maintenance in Phase 1

The basic BICP algorithm presented in Algorithm 2 potentially saves significant amount of time in its Phase 1 by avoiding re-decomposition of sub-tensors that have not seen any updates. Nevertheless, re-decomposing even only the updated sub-tensors from scratch can be expensive. Therefore, when updates to the sub-tensors are small, we can instead revise the existing sub-factors directly, rather than re-decomposing the corresponding sub-tensors.

For this purpose, we note that the factor matrices of a tensor $\boldsymbol{\mathcal{Y}}$ can be considered as factor matrices of $\boldsymbol{\mathcal{Y}}$'s matricizations (the HOSVD [6] algorithm relies on this observation to decompose a given tensor to its factors). Consequently, we can maintain the sub-factors by leveraging incremental matrix decomposition algorithms, like SPIRIT [27] or LWI-SVD [8]. Given a new row vector, x, SPIRIT first finds x's projection y, on the space defined by the current factor $\boldsymbol{U}$, by projecting x onto $\boldsymbol{U}$. Given this projection, an energy matrix, $\mathcal{S}$, which describes how much of the energy of the original matrix is captured by each column of the factor matrix, and a "forgetting factor" to control the speed with

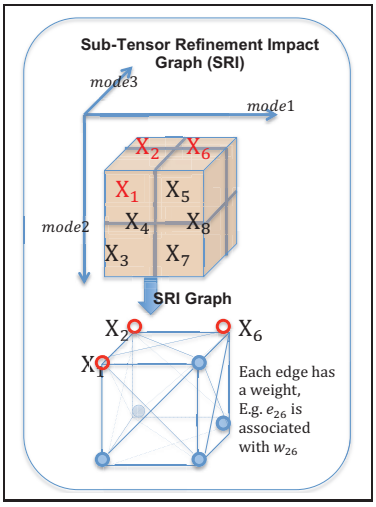


Figure 4: Illustration of the sub-tensor refinement impact (SRI) graph construction

which the factor is revised, it then revises the factor, $\boldsymbol{U}$ in a way that accounts best for y. Intuitively, the larger the error between the new row and its description by the old factor matrix $\boldsymbol{U}$, the more $\boldsymbol{U}$ is revised.

The STA [32] algorithm applies this idea to maintain factors of an evolving tensor. In this paper, however, we note that tracking the factor matrices of the whole tensor can have negative effects on accuracy and efficiency. In particular, as we experimentally validate in Section 5, tracking the factors of the whole tensor introduces unnecessary reductions in accuracy. In contrast, maintaining factor matrices of an updated sub-tensor by applying the tracking process on the matricizations of that sub-tensor (as shown in Algorithm 3) significantly boosts accuracy. Moreover, as we also experimentally validate in Section 5, focusing only on matricization columns that carry most of the energy of the update matrix, Δ_t, (see Steps 1 and 2c of Algorithm 3), not only significantly improves execution time, but can also significantly improve accuracy, when updates are clustered (e.g. on a single fiber of the sub-tensor).

4.2 Optimization #2- Reducing Redundant Refinements in Phase 2

The basic BICP method presented in Algorithm 2 potentially saves significant amount of time in its Phase 2 by focusing the refinements on the factors that are directly relevant to the updated sub-tensor. In this section, we provide several optimizations that help achieve even higher efficiencies, while maintaining high accuracy.

As we have discussed earlier in Section 4, 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. However, since the refinement process is iterative, sub-tensors that are not directly related to the updated sub-tensor may also become affected during the further stages of 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.

4.2.1 *Update Sensitive Refinement*

Given an update, Δ on tensor $\boldsymbol{\mathcal{X}}$, our second BICP optimization assigns an update-sensitive impact score, $I_\Delta(\boldsymbol{\mathcal{X}}_{\vec{k}})$,

to each sub-tensor, $\boldsymbol{\mathcal{X}}_{\vec{k}}$, and leverages this impact score to regulate the refinement process to eliminate redundant work:

- *Optimization 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 2-P:* While optimization 2-I can potentially save a lot of redundant work, completely ignoring low-impact tensors may be somewhat drastic. One alternative, which does not have an as drastic an impact as 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 assigned 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 2-R:* Note that optimization 2-I and 2-P are only applicable to sub-tensors with very low impact scores. For the rest of the sub-tensors, we need other optimizations to reduce refinement cost. One way to achieve this is to assign different ranks to high-impact sub-tensors according to their impact scores: naturally, the higher the target rank is, the more accurate the decomposition of the sub-tensor. Therefore, we assign lower ranks to the sub-tensors with relatively lower impact scores to save work, while maintaining accuracy (as verified in Section 5). We achieve this by adjusting the decomposition rank, $F_{\vec{k}}$ of $\boldsymbol{\mathcal{X}}_{\vec{k}}$, as a function of the corresponding tensor's update sensitive impact score:

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

  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)[2] based on their impacts scores. Once the new ranks are computed, we obtain new $U_{\vec{(k)}}$ factors with partial ranks $F_{\vec{k}}$ for $\mathcal{X}_{\vec{k}}$ and refine these incrementally in Phase 2.

Next, we discuss how to compute the impact scores used by the three optimization strategies presented above.

4.2.2 Computing Sub-Tensor Impact Scores Relative to the Tensor Updates

Let us be given an input tensor, $\boldsymbol{\mathcal{X}}$, an update, Δ, in this tensor, and a partitioning pattern, $\mathcal{K}$, that splits $\boldsymbol{\mathcal{X}}$ into sub-tensors. The optimized BICP algorithm, BICP_{opt}, needs to associate an impact score, $I_\Delta(\boldsymbol{\mathcal{X}}_{\vec{k}})$, to each sub-tensor, $\boldsymbol{\mathcal{X}}_{\vec{k}}$, where $\vec{k} \in \mathcal{K}$, that regulates the refinement process. For this purpose, we first construct a sub-tensor refinement impact (SRI) graph that reflects how refinements in each sub-tensor impact other sub-tensors in Phase 2.

[2]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.

Sub-Tensor Refinement Impact (SRI) Graph.

The key goal of the SRI graph is to account for propagation of refinements along the tensor during the refinement process in Phase 2. Let $\boldsymbol{\mathcal{X}}$ be a tensor partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ as specified in Section 3.3. The corresponding SRI graph is a directed, weighted graph, $G(V, E, w())$, where

- for each $\boldsymbol{\mathcal{X}}_{\vec{k}} \in \boldsymbol{\mathcal{X}}$, there exists a corresponding $v_{\vec{k}} \in V$,
- for each $\boldsymbol{\mathcal{X}}_{\vec{l}} \in direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{k}})$, there exists a directed edge $v_{\vec{l}} \rightarrow v_{\vec{k}}$ in E (see Section 4), and
- $w()$ is a weight function, where $w(v_{\vec{l}} \rightarrow v_{\vec{k}})$ quantifies the impact of the decomposition of $\boldsymbol{\mathcal{X}}_{\vec{l}}$ on the decomposition of $\boldsymbol{\mathcal{X}}_{\vec{k}}$.

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; in contrast, if they are dissimilar, their impacts on each other will also be minimal. In other words, the weight of the edge from $v_{\vec{j}}$ to $v_{\vec{l}}$ should reflect the *alignment* between the sub-tensors $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$. More formally, let $\boldsymbol{\mathcal{X}}$ be a tensor partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$. Let also $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$ be two sub-tensors in $\mathfrak{X}$, such that

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

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*, $align(\boldsymbol{\mathcal{X}}_{\vec{j}}, \boldsymbol{\mathcal{X}}_{\vec{l}}, A)$, between $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$ as

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

where $cos()$ is the cosine similarity function and the vector $\vec{\boldsymbol{c}}_{\vec{j}}(A)$ captures the value distribution of the tensor $\boldsymbol{\mathcal{X}}_{\vec{j}}$ along the modes in A which compresses the tensor to a matrix along mode A by calculating the standard Frobenius norm. Given this, we set the edge weights of the edge $(v_{\vec{j}} \rightarrow v_{\vec{l}}) \in E$ in the sub-tensor impact graph as follows:

$$w_1(v_{\vec{j}} \rightarrow v_{\vec{l}}) = \frac{align(\boldsymbol{\mathcal{X}}_{\vec{j}}, \boldsymbol{\mathcal{X}}_{\vec{l}})}{\sum_{(v_{\vec{j}} \rightarrow v_{\vec{m}}) \in E} align(\boldsymbol{\mathcal{X}}_{\vec{j}}, \boldsymbol{\mathcal{X}}_{\vec{m}})}.$$

EXAMPLE 1. *We visualize this in Figure 4. Here, tensor $\boldsymbol{\mathcal{X}}$ is partitioned into eight sub-tensors $\boldsymbol{\mathcal{X}}_1$ to $\boldsymbol{\mathcal{X}}_8$, and sub-tensors $\boldsymbol{\mathcal{X}}_1$, $\boldsymbol{\mathcal{X}}_2$ and $\boldsymbol{\mathcal{X}}_6$ contain the update to the tensor.*

- *Firstly, we build the Sub-Tensor refinement impact graph and assign weight using equation in Section 4.2.2.*
- *Then, we calculate the impact score using Personalized PageRank(PPR) Equation introduced in Section 4.2.2. When calculating PPR score, seed set are the nodes of updated sub-tensors: $\boldsymbol{\mathcal{X}}_1$, $\boldsymbol{\mathcal{X}}_2$, and $\boldsymbol{\mathcal{X}}_6$.*
- *After we obtain the impact scores, we assign the decomposition ranks according to their impact score using the method in Section 4.2* Optimization 2-R.

Computing the Refinement Impact Scores.

The edges on the sub-tensor refinement impact (SRI) graph, G, describe the refinement interdependencies among the sub-tensors, relative to the given update, Δ, on tensor, $\boldsymbol{\mathcal{X}}$ sub-tensors. We leverage this graph to measure how

Algorithm 4 Optimized block-centric incremental tensor decomposition

Input: original tensor, $\boldsymbol{\mathcal{X}}$; tensor block partitioning pattern, $\mathcal{K}$; rank, F, block decomposition, $\mathring{\boldsymbol{\mathcal{X}}} = \langle U, P, Q, A\rangle$, of $\boldsymbol{\mathcal{X}}$; a tensor update, Δ; and optimization parameters: optimization strategy (2-I, 2-P, or 2-R), the percentage, $L\%$ of the low impact sub-tensors for which refinement work is reduced, and for strategy 2-P, the probability, p_{low}, of updates for low impact sub-tensors.

Output: CP tensor decomposition $\mathring{\boldsymbol{\mathcal{X}}}_\Delta$ of the updated tensor

1. Let $\mathcal{T}$ be the set of sub-tensors containing the update Δ
2. *Phase 1:* for all $\vec{t} \in \mathcal{T}$
   - Apply tensor tracking method Algorithm 3 for $\vec{t}$
3. *Phase 2:*
   (a) $\mathcal{R} = \bigcup_{\vec{t}\in\mathcal{T}}$ *direct_impact*$(\boldsymbol{\mathcal{X}}_{\vec{t}})$
   (b) Get refinement impact score, $I_\Delta(\vec{r})$, for all $\vec{r} \in \mathcal{R}$
   (c) For the lowest $L\%$ of sub-tensors $\vec{r} \in \mathcal{R}$,
      i. $F_{\vec{r}} = F$
      ii. if 2-I or 2-R, then $p_{\vec{r}} = 0.0$
      iii. if 2-P, then $p_{\vec{r}} = p_{low}$
   (d) For the highest $(100 - L)\%$ of sub-tensors in $\vec{r} \in \mathcal{R}$,
      i. $p_{\vec{r}} = 1.0$
      ii. if 2-I or 2-P, then $F_{\vec{r}} = F$
      iii. if 2-R, then
         - $F_{\vec{r}} = F \times \frac{PPR(\boldsymbol{\mathcal{X}}_{\vec{r}})}{max\{PPR(\mathcal{R})\}}$
         - Truncate $U_{\vec{r}}$ and $P_{\vec{r}}$ according to $F_{\vec{r}}$
   (e) Repeat for $\vec{r} \in \mathcal{R}$
      - With probability $p_{\vec{r}}$ do
        i. for each mode $1 \le m \le N$ for which $\vec{r}$ is aligned with any $\vec{t} \in \mathcal{T}$
           A. update $\boldsymbol{A}^{(m)}_{(r_m)}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$, for each block $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,r_m,*,\ldots,*]} \in \mathcal{R}$; more specifically,
              – compute $\boldsymbol{T}^{(m)}_{(r_m)}$, which involves the use of $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$ (i.e. the mode-i factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,r_m,*,\ldots,*]}$)
              – revise $\boldsymbol{P}_{[*,\ldots,*,r_m,*,\ldots,*]}$ and $\boldsymbol{Q}_{[*,\ldots,*,r_m,*,\ldots,*]}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,r_m,*,\ldots,*]}$ and $\boldsymbol{A}^{(m)}_{(r_m)}$
              – compute $\boldsymbol{S}^{(m)}_{(r_m)}$
              – update $\boldsymbol{A}^{(m)}_{(r_m)}$
           B. for all $\vec{l} = [*, \ldots, *, r_i, *, \ldots, *] \in \mathcal{R}$
              – update $\boldsymbol{P}_{\vec{l}}$ and $\boldsymbol{Q}_{\vec{l}}$ using
                * $\boldsymbol{U}^{(m)}_{\vec{l}}$ and $\boldsymbol{A}^{(m)}_{(r_m)}$

   until stopping condition
4. Return $\mathring{\boldsymbol{\mathcal{X}}}_\Delta$

refinements propagate within G including both *direct* and *indirect* refinements. In particular, we rely on personalized PageRank (PPR [5, 9]) 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. PPR complements this with a seed node set, $S \subseteq V$, which serves as the context: 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). In particular, the PPR score $\vec{p}[i]$, of v_i is obtained by solving

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

where $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\vec{s}$ is a re-seeding vector such that if $v_i \in S$, then $\vec{s}[i] = \frac{1}{\|S\|}$ and $\vec{s}[i] = 0$, otherwise. 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 scores.

We note that the iterative nature of the random-walk process underlying PPR fits well with how the effects of refinements propagate during the iterative ALS process. In particular, given an update Δ and the corresponding set, $\mathcal{T}$, of updated sub-tensors, we set the seed vector $\vec{s}$ such that the non-zero entries correspond to sub-tensors in $\mathcal{T}$ and solve the above equation for vector $\vec{p}$. Given this, we then compute the impact score, $I_\Delta(\boldsymbol{\mathcal{X}}_{\vec{k}})$ as $I_\Delta(\boldsymbol{\mathcal{X}}_{\vec{k}}) = \vec{p}[\vec{k}]$. 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 compute impact scores is negligible next to the cost of tensor decomposition.

4.3 Optimized BICP Algorithm, BICP$_{opt}$

Algorithm 4 presents the optimized BICP algorithm, which combines the optimizations presented above. In the next section, we experimentally investigate the performances of these optimizations.

5. EXPERIMENTS

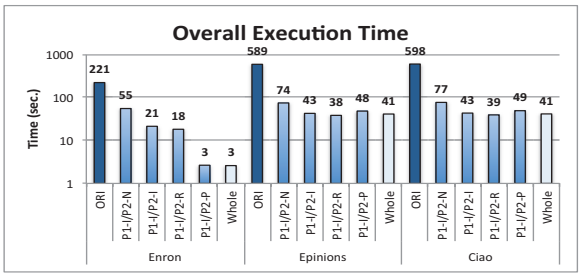
In this section, we report experiment results to assess the efficiency and effectiveness of the proposed two-phase block-centeric approach to incremental tensor decomposition.

5.1 Experiment Setup

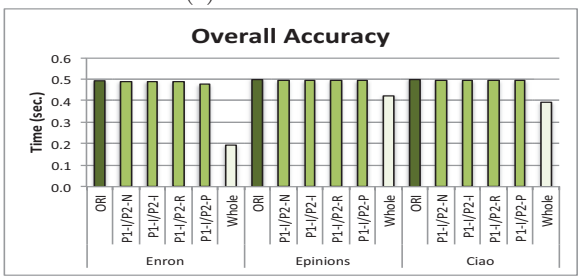
Data Sets. In these experiments, we used three datasets: *Epinions* [37], *Ciao* [37], and *Enron* [28]. The first two of these are comparable in terms of their sizes and semantics: they are both $5000 \times 5000 \times 27$ tensors, with schema $\langle user, item, category\rangle$, and densities 1.089×10^{-6} 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 considered both *clustered* and *distributed* updates: (a) For *clustered updates*, we divided the tensor into 64 blocks (using $4 \times 4 \times 4$ partitioning) and applied all the updates to one of these blocks; (b) in the case of *distributed updates*, we varied the percentage, B, of blocks that are updated: $B \sim 5\%$ (4 blocks – default), $\sim 10\%$ (7 blocks), and $\sim 20\%$ (13 blocks). Once the blocks are selected, we randomly pick a slice on the block and update $C = 10\%$ (default) to 30% of the fibers on this slice.

Alternative Strategies. In this section, we consider the following strategies to maintain the tensor decomposition: (a) the first approach (`ORI`) is to apply BICP with the basic CPALS algorithm [25] for Phase 1 and the original block-centric iterative refinement tensor decomposition process without utilizing any incremental method for Phase 2. In addition to this, we considered several optimizations to BCIP. For Phase 1, there are two options: (b) to redo the tensor decomposition for the updated sub-tensors (`P1N`) or (c) to utilize incremental tracking algorithm (`P1I`, default). For Phase 2, again, we have several alternatives: (d) applying Phase 2 without any refinement impact score based optimization (`P2N`), (e) ignoring $L\%$ of sub-tensors with the lowest impact scores (`P2I`), and (f) reducing the de-



(a) Execution times



(b) Decomposition accuracies

Figure 5: Comparison of (a) execution times and (b) accuracies under the default configuration: the proposed techniques provide several orders of gain in execution time relative to `ORI`, while (unlike `Whole`, they match `ORI`'s accuracy

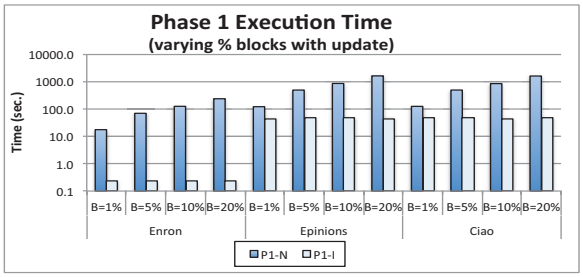
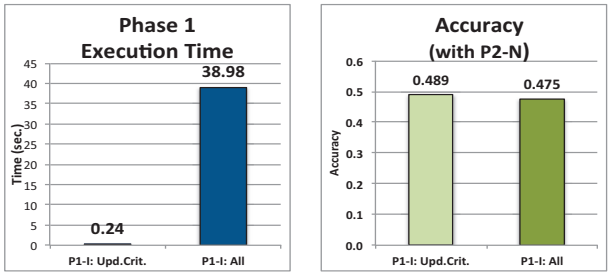


Figure 6: Phase 1 execution time, with and without incremental factor tracking (i.e., `P1N` vs `P1I`), as a function of the percent of blocks with updates



(a) Exec. time (b) Accuracy

Figure 7: (a) Execution time and (b) accuracy – with and without update-critical vector maintenance in Phase 1 (Enron Data)

composition rank of sub-tensors (`P2R`), or (g) using probabilistic refinements for sub-tensors with low impact scores (`P2P`). In these experiments, we varied L between 10% and 75% (with default set to $L = 50\%$) and, for `P2P`, we varied the update probability between 0.0 and 1.0, with the default set to $p = 0.1$. (h) In addition to the optimization of BCIP, we also applied incremental factor tracking (Algorithm 3) to the whole tensor, as in STA [32] – in the charts, we refer to this approach as `Whole`.

Evaluation Criteria. We use the measure reported in Section 3.2 to assess decomposition accuracy. We also report decomposition time (Phase 1, Phase 2, and total) 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. [3].

5.2 Discussions of the Results

We now report the results of the experiments outlined above and present our interpretations of these results.

5.2.1 General Overview

Figure 5 compares execution times and accuracies of several approaches. Here, `ORI` indicates the non-optimized BCIP, whereas `Whole` indicates application of factor tracking to the whole tensor. The other four techniques in the figure (`P1I/P2N`, `P1I/P2I`, `P1I/P2R`, `P1I/P2P` all correspond to different optimizations of the proposed BICP approach, with incremental factor tracking, `P1I`, in Phase 1 and four different strategies (`P2N`, `P2I`, `P2R`, `P2P`) for Phase 2.

Firstly, a quick look at 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.

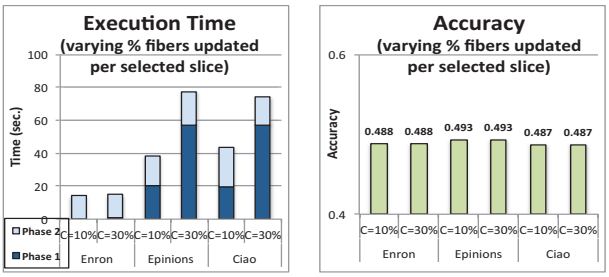
They key observation in Figure 5 is that the various optimizations of BCIP provide several orders of gain in execution time while matching the accuracy of non-optimized version almost perfectly (i.e., the optimizations come without significant quality penalties). In contrast, the alternative strategy, `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 next study the impact of the proposed BCIP optimizations in greater detail.

5.2.2 Evaluation of Phase-1 Optimizations

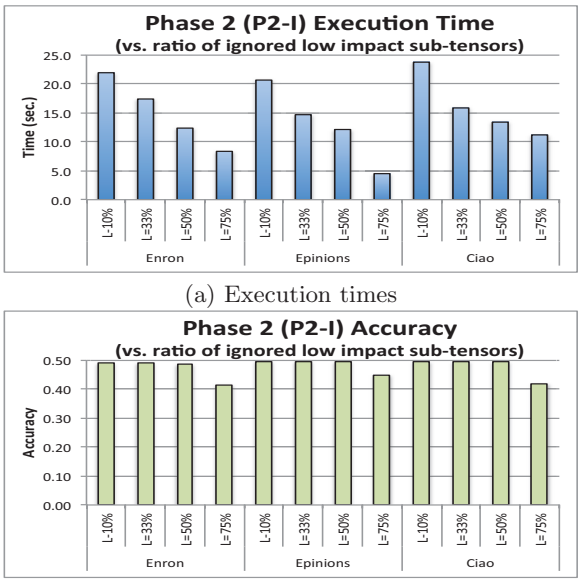
Figure 6 plots the Phase 1 execution time of BICP, with and without incremental factor tracking, as a function of the percent of blocks with updates. As we see in this figure, the incremental sub-factor tracking algorithm (Algorithm 3) provides significant gains in the execution time of Phase 1, especially for the denser, Enron, data set. Moreover, with Algorithm 3, the overall cost of Phase 1 stays more or less constant independent of the number of blocks being tracked, indicating that the process itself is very efficient and the major cost is the time to set up the relevant data structures in constant time.

Note that, unlike STA [32], Algorithm 3 does not consider in factor maintenance all modes and all effected fibers in all matricizations. Instead, it focuses on fibers that are update-critical (with highest energy among the all the effected fibers). As we see in Figure 7, this not only provides significant gains in execution time, it actually leads to a slightly better overall accuracy: since the factor matrix tracking algorithm is imperfect, revising the factor matri-



(a) Exec. time (b) Accuracy

Figure 8: (a) Execution time and (b) accuracy as the ratio of the modified fibers vary



(a) Execution times

(b) Decomposition accuracies

Figure 9: Phase 2 (a) execution time and (b) accuracy for P2I, when varying the ratio of the low impact sub-tensors ignored during refinement

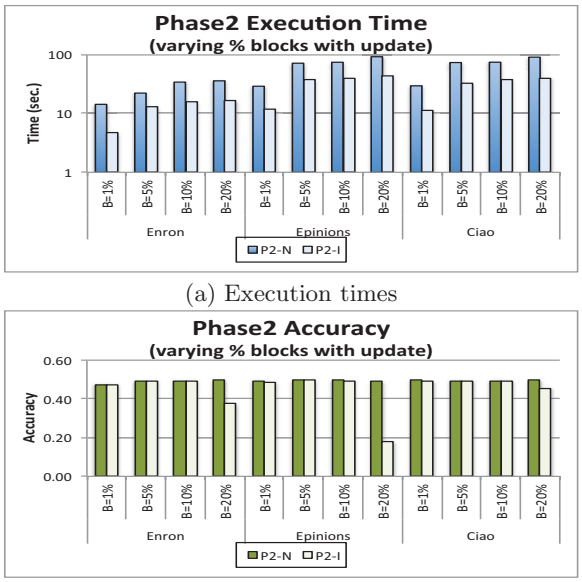
ces for modified fibers with relatively low energy of change potentially adds errors, rather than reducing them.

Figure 8 illustrates the effect of varying the number of fibers modified at each update: since Algorithm 3 in Phase 1 focuses the work on fibers with large changes, Phase 1 cost is directly proportional to the number of updated fibers. In contrast, neither the execution time of Phase 2, nor the accuracy of the overall decomposition process is affected by the number of updated fibers,

5.2.3 *Evaluation of Phase-2 Optimizations*

One of the optimizations, P2I, for Phase 2 involves identifying and ignoring low impact sub-tensors during the refinement process. In Figure 5, we had seen that P2I provides significant gains over the non-optimized Phase 2, P2N. As we see in Figure 9, ignoring such low impact sub-tensors can indeed save significant amounts of time. Moreover, unless the ratio of the ignored tensors is very high (75%, in the considered default scenario), these gains are obtained without any accuracy penalties.

Figure 10 further confirms the above results. The charts in this figure show the Phase 2 execution time and the resulting overall accuracy as functions of the ratio of the blocks with updates, with and without the ignoring of low impact



(a) Execution times

(b) Decomposition accuracies

Figure 10: Phase 2 (a) execution time and (b) accuracy, with and without low-impact sub-tensor ignoring (P2I vs P2N), as a function of the percent of blocks with updates

Table 1: Impact of different low-impact sub-tensor refinement probabilities in P2P (for Enron)

	Refinement prob.		
	0.0	0.1	1.0
Phase 2 Time (sec.)	11.75	3.46	20.59
Accuracy	0.49	0.42	0.49
#Iterations	827.38	85.00	959.77
#Updates per Iter.	79.49	85.93	132.30

sub-tensors (i.e, P2I vs P2N). As we see in the figure, P2I provides significant execution time gains over P2N at almost no accuracy cost, except when the number of updated sub-tensors reaches 20% of all blocks: when a large number of sub-tensors are updated, more of the remaining sub-tensors become relevant and dropping a large ratio (default, 50%), of low impact score sub-tensors from consideration during refinement becomes counter-productive. Therefore, we propose to address this in our future work by introducing an adaptive impact cut-off rather than a imposing fixed ratio of ignored blocks.

To see the impacts of the P2P and P2R optimization, we again consider Figure 5: in this figure, we see that P2R, which adjusts the ranks of high impact sub-tensors, provides additional execution time gains over P2I: for the two social media data Epinions and Ciao, P2R provides the best execution time, with no accuracy penalty.

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. One interesting result, however, is that for the Enron data P2P approach, which increases the number of refinements relative to P2I, leads to a significant reduction in execution time, albeit at a slight accuracy penalty. We see the reason for this in Table 1: here

0.0 corresponds to ignoring all low-impact sub-tensors (i.e., `P2I`), whereas 1.0 corresponds to not ignoring any sub-tensor (i.e., `P2N`). As we see here, in Enron data, the introduction of a refinement probability different from 0.0 and 0.1 significantly reduces the number of refinement iterations required for Phase 2's convergence – thereby requiring significantly lesser time, but also introducing higher error. 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. Instead, we recommend the users to leverage the `P2R` optimization, which provides execution time gains, without any reduction in accuracy.

6. CONCLUSIONS

Computational complexity of tensor decomposition is a major bottleneck in many applications. Especially when the analysis results need to be incrementally maintained, re-decomposition of the whole tensor with each and every update will incur high computational costs. In this paper, we introduce a two-phase block-based incremental CP tensor decomposition (BICP) approach: In the first phase, BICP only revises the decompositions of the tensors that contain updated data. Moreover, when updates are relatively small with respect to the block size, BICP relies on a incremental factor tracking to avoid re-decomposition of the updated sub-tensor. In the second phase, BICP limits the refinement process to only those blocks that are aligned with the updated block and utilizes an automatically computed refinement impact score to eliminate unnecessary refinement of sub-tensors. Experiment results on real datasets show that BICP can significantly reduce computational cost of obtaining the decomposition of the updated tensor, while maintaining high accuracy.

7. REFERENCES

[1] E. Acar, *et al.* Multiway analysis of epilepsy tensors. Bioinformatics, pp. 10-18, 2007.

[2] C. A. Andersson and R. Bro. The n-way toolbox for matlab. Chemometrics and Intelligent Laboratory Systems, 52(1):1-4, National Labs, 2000.

[3] B. W. Bader, T. G. Kolda, *et al.* MATLAB Tensor Toolbox Version 2.5, Available online, January 2012. URL: http://www.sandia.gov/∼tgkolda/TensorToolbox.

[4] A. Beutel, *et al.* Flexifact: Scalable flexible factorization of coupled tensors on Hadoop. SDM, 2014.

[5] A. Balmin, *et al.* ObjectRank: Authority-based keyword search in databases. VLDB, 2004.

[6] P.Baranyi, *et al.* Definition of the HOSVD based canonical form of polytopic dynamic models. IEEE International Conference on Mechatronics, Pages 660-665. 2006.

[7] J. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 1970.

[8] X.Chen and K. S. Candan. LWI-SVD: low-rank, windowed, incremental singular value decompositions on time-evolving data sets. KDD, 2014.

[9] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. WWW, 2007.

[10] J. H. Choi and S. V. Dfacto. Distributed factorization of tensors. Advances in Neural Information Processing Systems 27, pages 1296-1304, 2014.

[11] I. Davidson, *et al.* Network discovery via constrained tensor analysis of fMRI data. SIGKDD, pages 194-202. 2013.

[12] R. A. Harshman, Foundations of the PARAFAC procedure: Model and conditions for an explanatory multi-mode factor analysis. UCLA Working Papers in Phonetics, 16:1-84, 1970.

[13] I. Jeon, *et al.* HaTen2: Billion-scale tensor decompositions. ICDE'15, 2015.

[14] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos. Gigatensor: scaling tensor analysis up by 100 times algorithms and discoveries. KDD, 2012

[15] M. Kim and K.S. Candan. Efficient Static and Dynamic In-Database Tensor Decompositions on Chunk-Based Array Stores. CIKM, 2014.

[16] T.G. Kolda and B.W. Bader. The tophits model for higher-order web link analysis. Workshop on Link Analysis, Counterterrorism and Security, 2006

[17] T. G. Kolda, J. Sun. Scalable tensor decompositions for multi-aspect data mining. ICDM, 2008.

[18] W.Hu, X.Li, X.Zhang, X.Shi, S.Maybank, and Z.Zhang, Incremental Tensor Subspace Learning and Its Application to Foreground Segmentation and Tracking, Int. J.Comput. Vis.(2001) 91:303-327

[19] X. Li, S. Huang, K.S. Candan, M.L. Sapino. Focusing Decomposition Accuracy by Personalizing Tensor Decomposition (PTD). CIKM'14, 2014.

[20] X. Li, S. Huang, K.S. Candan, M.L. Sapino. 2PCP: Two-Phase CP Decomposition for Billion-Scale Dense Tensors. ICDE'16. 2016.

[21] X.Li, W.Hu, Z.Zhang, and G.Luo, Robust Visual Tracking Based on Incremental Tensor Subspace Learning. IEEE 11th Int. Conf. on Comput. Vis.(2007) pp. 1-8

[22] X.Ma, *et al.* Dynamic Updateing and Downdationg Matrix SVD and tensor HOSVD for adaptive indexing and Retrieval of Motion Trajectories, ICASSP, 2009.

[23] D.Nion, and N. D. Sidiropoulos, Adaptive Algorithms to Track the PARAFAC Decomposition of a Third-Order Tensor, IEEE Trans on Sig. Proc. 57-6(2009), pp. 2299-2310

[24] A. H. Phan et al. CANDECOMP/PARAFAC decomposition of high-order tensors through tensor reshaping. *TSP*, 2013.

[25] A.H. Phan and A. Cichocki. PARAFAC algorithms for large-scale problems, Neurocomputing, 74(11), 2011.

[26] E.Papalexakis, C.Faloutsos, N.Sidiropoulos.Parcube: Sparse parallelizable tensor decompositions. ECML/PKDD. 2012.

[27] S.Papadimitriou, J.Sun, C. Faloutsos. Streaming pattern discovery in multiple time-series. VLDB '05. 2015.

[28] C. E. Priebe, *et al.* Enron data set, 2006. http://cis.jhu.edu/ parky/Enron/enron.html

[29] A.Sobral, *et al.* Incremental and Multi-feature Tensor Subspace Learning Applied for background Modeling and Subtraction, ICIAR'14, 2014.

[30] J. Sun, S. Papadimitriou, and P. S. Yu. Window based tensor analysis on high dimensional and multi aspect streams. ICDM, pages 1076-1080, 2006.

[31] E. Papalexakis, C. Faloutsos, N.Sidiropoulos. Parcube: Sparse parallelizable CANDECOMP-PARAFAC tensor decompositions. TKDD 10(1): 3. 2015.

[32] Sun, J., Tao, D., Papadimitriou, S., Yu, P.S., Faloutsos, C.: Incremental tensor analysis: Theory and applications. ACM Trans. Knowl. Discov. Data **2**(3) (October 2008)

[33] K. Shin and U. Kang. Distributed methods for high-dimensional and large-scale tensor factorization. ICDM, pages 989–994, 2014.

[34] J.T. Sun, H.J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: a novel approach to personalized web search. WWW, 2005

[35] C. E. Tsourakakis, Mach: Fast randomized tensor decompositions. Arxiv preprint arXiv:0909.4969, 2009

[36] L. Tucker, Some mathematical notes on three-mode factor analysis. Psychometrika, 31:279-311, 1966.

[37] http://www.public.asu.edu/∼jtang20/datasetcode/truststudy.htm