Optimization of Block-based Tensor Decompositions through Sub-Tensor Impact Graphs and Applications to Dynamicity in Data and User Focus

by

Shengyu Huang

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2021 by the
Graduate Supervisory Committee:

K. Selçuk Candan, Chair
Hasan Davulcu
Maria Luisa Sapino
Hanghang Tong
Jia Zou

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

Tensors are commonly used for representing multi-dimensional data, such as Web graphs, sensor streams, and social networks. As a consequence of the increase in the use of tensors, tensor decomposition operations began to form the basis for many data analysis and knowledge discovery tasks, from clustering, trend detection, anomaly detection to correlation analysis [31, 38].

It is well known that Singular Value matrix Decomposition (SVD) [9] is used to extract latent semantics for matrix data. When apply svd to tensors, which have more than two modes, it is tensor decomposition. The two most popular tensor decomposition algorithms are the Tucker [54] and the CP [19] decompositions. Intuitively, they both generalize SVD to tensors. However, one key problem with tensor decomposition is its computational complexity which may cause system bottleneck. Therefore, two phase block-centric CP tensor decomposition (2PCP) was proposed to partition the tensor into small sub-tensors, execute sub-tensor decomposition in parallel and combine the factors from each sub-tensor into final decomposition factors through iterative refinement process.

Consequently, I proposed Sub-tensor Impact Graph (SIG) to account for inaccuracy propagation among sub-tensors and measure the impact of decomposition of sub-tensors on the other's decomposition, Based on SIG, I proposed several optimization strategies to optimize 2PCP's phase-2 refinement process. Furthermore, I applied SIG and optimization strategies for data focus, data evolution and focus shifting in tensor analysis. Personalized Tensor Decomposition (PTD) is proposed to account for the users focus given the observations that in many 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. PTD takes as input one or more areas of focus and performs the decomposition in such a way that, when reconstructed, the accuracy of the tensor is boosted for these areas of focus.

A related challenge of data evolution in tensor analytics is incremental tensor decomposition since re-computation 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. To avoid re-decomposition, I 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.

I further extend the research focus on user focus shift. User focus may change over time as data is evolving along the time. Although PTD is efficient, re-computation for each user preference update can be bottleneck for the system. Therefore I propose dynamic evolving user focus tensor decomposition which can smartly reuse the existing decomposition result to improve the efficiency of evolving user focus block decomposition.

DEDICATION

To my family

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. K. Selçuk Candan. Without his mentorship and endless support, I could not finish this journey. He gives me an example of an excellent researcher, mentor, instructor. His feedback and insights were extremely critical to achieving my goal, and I will always be grateful to him for that.

I would like to thank my committee members, Prof. Hasan Davulcu, Prof. Maria Luisa Sapino, Prof. Hanghang Tong and Prof. Jia Zou. Thanks for all your guidance through this process. I would also like to thank the graduate advisors and technical staff members at CIDSE for always being very helpful.

I would like to thank all my excellent lab members of EmitLab, Parth Nagarkar, Jung Hyun Kim, Xilun Chen, Mijung Kim, Mithila Nagendra, Yash Garg, Xinsheng Li, Sicong Liu, Silvestro Poccia, Ashish Gadkari, Mao-Lin Li, and Hans Behrens. I am very happy to have this opportunity to work with such wonderful team members.

Lastly, I would especially like to thank my amazing family for the love, support, and constant encouragement I have gotten over the years. Without your support, I definitely could not finish my thesis work.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Due to the data explosion, we are entering a big data generation. Large-scale data sets bring us both the challenges and knowledges. How to efficiently and effectively make use of these data sets become popular research problems. Traditional 2D representation of data cannot satisfy the needs of describing multi-dimensional data. Tensor is a multi-dimensional array and usually represents multi-dimensional data such as web graphs and social networks.

We can think tensor as a high-order generalization of a matrix. Since matrices only have representations for 2D, there are many scenarios that may need more: time series data where data sets are evolving as time goes(e.g., authors, keywords, timestamps) and image(or video) data analysis. Two-way analysis methods may not capture underlying information of the data especially when data sets have multiple relations and two-way factor models are often not unique. To obtain a unique solution, SVD (Singular Value Decomposition) of a matrix requires additional constraints such as orthogonality constraints, on the other hand, CP [19] decomposition(one of the most popular tensor decomposition) is unique with much weaker conditions [32]. Tensor-based data analysis has advantage over two-way data model in numerous research areas in terms of uniqueness, robustness to noise, and easy interpretation, etc.

Tensor decomposition can be regarded as a generalization of matrix(2D) decomposition to multi-dimensional data. The most two popular tensor decompositions are CANDECOMP [13] and PARAFAC [19] decompositions (known as the CP decomposition), and Tucker decomposition [54]. Figure 1.1 gives an example of tensor decomposition: 4-mode tensor represents that user with feature genre gives a rating score to the movie, and this tensor is decomposed

with CP method into independent components.

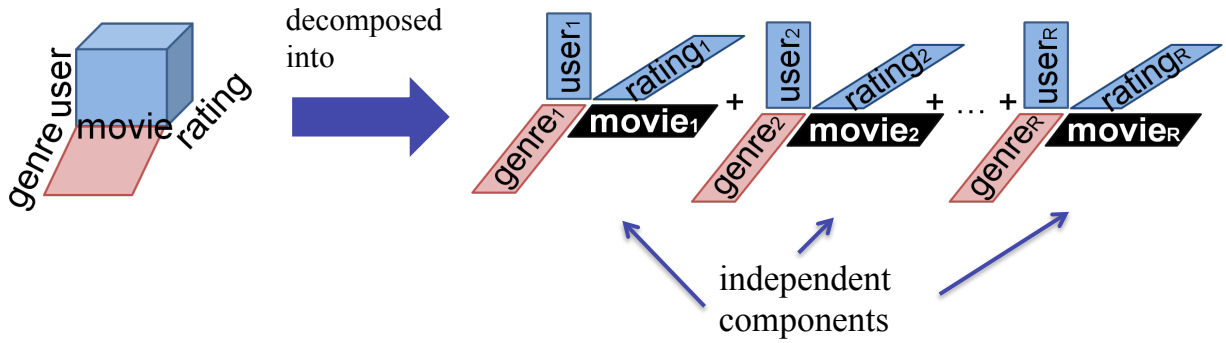


Figure 1.1: 4-Mode Tensor Decomposition with CP

Tensor decomposition is widely used for multi-aspect data analysis for multidimensional data. These operations form the basis for many data analysis and knowledge discovery tasks. However the bottleneck of tensor decomposition is their computational complexity because tensor decomposition process results in dense (and hence large) intermediary data, even when the input tensor is sparse (and hence small) and this is known as the intermediate memory blow-up problem [25, 24]. To deal with data explosion, several implementations of tensor decomposition operations on large scale data sets have been proposed. GridPARAFAC [39], 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. TensorDB [28, 29] leverages a block-based framework to store and retrieve data, extends array operations to tensor operations, and introduces optimization schemes for in-database tensor decomposition. HaTen2 [24] focuses on sparse tensors and presents a scalable tensor decomposition suite of methods for Tucker and PARAFAC decompositions on the MapReduce framework. However, with distributed computation, l/O costs is an inevitable problem as they need I/O to fetch data either from disk or from the network. It has been experimentally verified that I/O or communication overhead of iterative algorithms (especially on a distributed platform

like MapReduce) can be very expensive.

1.1 Sub-tensor Impact Graph

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 [39], 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.

While block-based tensor decomposition techniques [39, 28] 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*.

1.1.1 Contribution: Sub-tensor Impact Graph(SIG)

As described above, block-based tensor decomposition techniques (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 is that, during iterative refinement process, the

various sub-tensor decompositions interact with each other and any inaccuracies in individual sub-tensor decompositions can propagate through the update rules 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, I introduce the notion of *sub-tensor impact graphs (SIGs)*, 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 such as incrementally maintain tensor decomposition in the presence of data evolution, efficiently conduct tensor decomposition when user only focus on part of the tensor and etc.

1.2 Robust User Focus Network

As shown in Section 1.1, tensor is partitioned into small sub-tensors and SIG is built to present the relation graph based on their connection information which naturally form a network. To optimize SIG application, we focus on PageRank to capture the potential network topology and measure the significance of the node to analyze the connections among nodes.

The well-known PageRank algorithm [11] associates an importance score to each node relying on random walks: Let us consider a weighted, directed graph $G(V, E)$, where the weight of the edge $e_j \in E$ is denoted as $w_j (\geq 0)$ and $\sum_{e_j \in outedge(v_i)} w_j = 1.0$. The PageRank score of nodes V is the stationary distribution of a random walk on G, denoted with a vector $\vec{p}$:

$$\vec{p} = (1-\beta)\mathbf{T}_G \times \vec{p} + \beta\vec{v}, \tag{1.1}$$

where $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\vec{v}$ is a so-called *teleportation* vector, where all entries are $\frac{1}{\|V\|}$. An early attempt to contextualize the PageRank scores was the *topic sensitive PageRank* [22] approach which adjusts the PageRank scores of the nodes by assigning the *teleportation* probabilities in vector $\vec{j}$ in a way that reflects the graph nodes' degrees of match to a given search topic. In many applications, however, the context relevant to the recommendation is defined not through an explicit query, but a subset of the nodes (often referred to as the "*seed nodes*") in the graph. Personalized PageRank (PPR) [36, 14], for example, takes into account a user's interest by modifying the teleportation vector taking into account a given set of *important* nodes which are the target of the random jumps: given a set of nodes $S \subseteq V$, instead of jumping to a random node in V with probability β, the random walk jumps to one of the nodes in the seed set, S, given by the user. More specifically, if we denote the *Personalized PageRank* (PPR) scores of the nodes in V with a vector $\vec{\pi}$, then

$$\vec{\pi} = (1-\beta)\mathbf{T}_G \times \vec{\pi} + \beta\vec{s}, \tag{1.2}$$

where $\vec{s}$ 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.

1.2.1 Contribution: Robust Personalized PageRank(RPR)

The above formulation of PPR assumes that all seeds are equally important in characterizing the users interest. This, however, may not always be the case, since in practice, the user feedback is often incomplete and noisy, the resulting rankings observed from PPR might be biased and might contain undesirable artifacts.

Therefore, I and my colleges proposed Robust Personalized PageRank (RPR) strategies, which are insensitive to noise in the set of seed nodes (and thus differentiate seeds well) and

in which the rankings are not overly biased towards the seed nodes. There are alternative methods proposed,here I introduce the core algorithm RPR-2. The main idea is to rely on a core seed set maximality principle that would tie the teleportation rates of the seeds to their contributions to the overall personalized PageRank score of the seed set. Therefore, RPR-2 score are computed as follows: Given a graph $G(V, E)$, a teleportation probability, β, and a seed set, S, the re-start vector $\vec{s}$ should be such that

$$\vec{\rho_2} = (1-\beta)\mathbf{T}_G\ \vec{\rho_2} + \beta\vec{s},$$

$$\sum_{v_i \in V} \vec{s}[i] = 1, \forall_{v_i \in V} 0 \leq \vec{s}[i] \leq 1, \sum_{v_i \in V} \vec{\rho_2}[i] = 1, 0 \leq \vec{\rho_2}[i] \leq 1,$$

and the following term is *maximized*:

$$seed\_set\_significance = \sum_{v_i \in S} \vec{\rho_2}[i].$$

However, maximization problem is potentially expensive. RPR converts the optimization problem to a set of linear equation that for every seed in the seed set, they will solve a single-seed PPR and calculate the summation of PPR value of all the seeds and we get the maximal as result. This is especially advantageous when graph is large as they can leverage any of the highly effective approximation algorithms[2, 7, 14, 17, 51] or parallelized implementations[8, 53] for computing these PPR scores. Besides, incremental calculation can be done very fast since each equation is calculated separately and is very easy to add or delete seed from calculation from sets of the linear equations.

1.3 Data Evolution in Tensor Analytics

As tensor becomes extremely large, re-computation of the whole tensor decomposition with each update will cause high computational costs and incur large memory overheads, especially in applications where data evolves over time and the tensor-based analysis results need to be continuously maintained. Therefore, Block-Incremental CP decomposition(BICP)

is proposed to solve the problem of re-computation of incremental tensor decomposition. BICP is extended from a block-centric alternative of tensor decomposition which explained in section 2.3: (a) in their first phase, they 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(Algorithm 2). BICP utilize the block-centric update architecture to better improve incremental tensor decomposition and extended with methods to eliminate waste and support reuse, which can provide an effective framework for incremental tensor analysis.

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. $\mathring{\boldsymbol{\mathcal{X}}}$. Based on the above observations, in this work, I 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 [41, 45] 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 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.

1.4 User Focus in Tensor Analytics

Due to the approximate nature of the tensors decomposition process, one way to reduce computational requirements might be to trade performance with accuracy. However, naturally, a drop in accuracy may not be acceptable in many applications. Therefore, this is not a feasible solution to tackle the computational cost. In many 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, the user may not be interested in maintaining high accuracy. For example, in a clustering application, the user might be interested in ensuring the clustering accuracy (i.e., differentiating power) for a high priority subset of the objects in the database.

As a specific example, consider a movie recommendation application, where tensor decomposition of a *movie-user-ratings* tensor is used for generating recommendations: in this example, the user may want to decompose this tensor in a way that maximizes accuracies for *recent movies* and the *most active users* of the system.

Relying on this observation, I propose a novel *Personalized Tensor Decomposition (PTD)* mechanism for accounting for the user's focus and interests during tensor decomposition: PTD takes as input (a) an input tensor and (b) user's interest in the form of one or more area of focus. Given this input, PTD performs the tensor decomposition operation in such a way that, when reconstructed, the accuracy of the decomposition is boosted within the high-priority areas of focus. The proposed algorithm partitions the given tensor based on the focus of the user and assigns different initial decomposition ranks for different partitions. One of the key challenges I addressed is to account for the impact of the initial decompositions of the partitions on the final accuracies of the high-priority partitions in the user's focus. In particular, in order to quantify how the inaccuracy in one sub-tensor impacts the accuracy of another sub-tensor, (a) PTD first constructs a sub-tensor impact (SI) graph and (b) analyzes this graph in the light of the user's interest to calculate initial decomposition ranks for the partitions of the tensor.

Intuitively, the proposed *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 in this paper 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*: for

example, the basic alternating least squares (ALS) based decomposition algorithms improve the accuracy of the whole tensor iteratively and it is not possible to separate accuracy of one part of the tensor from accuracies of other parts. Even when using block based decomposition approaches [39] (which partition the tensor into multiple blocks, decompose the blocks independently, and combine these initial decompositions) initial decomposition accuracy of one tensor partition may impact final decomposition accuracies of other partitions of the tensor.

Therefore, PTD is presented 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. Experimental results showed that PTD is very effective in boosting accuracy for high priority regions of the tensor, while helping reduce the overall tensor decomposition time.

1.5 User Focus Shift in Tensor Analytics

User focus may change over time as data is evolving along the time. Although PTD (introduce in Section 6) is efficient, re-computation for each user preference update can be bottleneck for the system. I think changing user's focus can be modeled in two scenarios. The first one is that the sub-tensors which users are interested in are changing smoothly; The second scenario is that users' focuses are changing from one set of sub-tensors to another set of sub-tensors. For both cases, I propose the method that is extended from PTD and BICP.

- The sub-tensors that users are interested in are changing smoothly. In this case, since the sub-tensor is updated, impact graph edges' weight need to be adjusted. Then new PPR scores need to be computed based on updated graph and ranks need to be re-assigned based on updated PPR scores.

- Users' focuses are changing smoothly (in this case, the sub-tensor itself is not updated): PTD obtains a better accuracy for the user's focus part while not losing accuracy, for the other parts by assigning different decomposition ranks to different part according to user's focus. If user's focus changes to another part, in this case, we can re-assign the ranks since the impact graph is already generated from previous step, here we only need to compute the Personalized PageRank(PPR) scores with updated seeds.

For both cases, if the updated rank is smaller than previous rank, then drop the columns from the factor matrices; if the updated rank is larger than previous rank, then this becomes an interesting problem that how to efficiently derive more columns for factor matrices from existing factors. The basic idea is to re-decompose the tensor with CP tensor decomposition. However, to improve re-decomposition efficiency, we reuse the existing decomposition result with appending random column as initialization.

Specifically, we propose several strategy to improve the overall (time and accuracy) efficiency in dealing with preference evolution in tensor analysis

- RR: Repartition, Re-decomposition: In phase-1, repartition the whole tensor based on new interest

- OR: Old partition, Re-decomposition: In phase-1, reuse old partition schema, but re-rank sub-tensors based on new interest, and re-decompose sub-tensors based on new rank

- OP: Old partition, Partial Re-decomposition: In phase-1, reuse old partition schema,

but re-rank sub-tensors based on new interest, and only redecompose subtensors with new rank larger than old rank. For the cases where new rank is smaller or equal to old rank, re-use decomposition result (truncate the sub-factor)

- OP*: Old partition, Partial Re-decomposition, Optimized initialization
  - In phase-1, reuse old partition schema, but re-rank sub-tensors based on new interest. For the cases where new rank is smaller or equal to old rank, re-use decomposition result (truncate the sub-factor)
  - For the cases where new rank is larger than old rank, we initiate the decomposition sub-factor with old decomposition result plus random initialization

1.6 Dissertation Outline

The dissertation is organized in the following ways:

- In Chapter 2, the background and related work about user focus and evolution challenge for tensor-based analysis is presented
- In Chapter 3, I proposed SIG to account for the propagation of inaccuracies along the tensor during a block-based decomposition process and presented several optimization strategies for phase-2 iterative refinement process of block-based tensor decomposition
- In Chapter 4, RPR is proposed to capture potential network topology by improving PPR with being insensitive to noise in seed nodes and rankings are more fairly distributed on seed sets.
- in Chapter 5, BICP is proposed, which leverages two-phase block-incremental CP-based tensor decomposition technique to efficiently and effectively maintains tensor decomposition results in the presence of dynamically evolving tensor data.

- In Chapter 6, I describes the algorithm of PTD to handle user focus challenge based tensor decomposition framework.
- In Chapter 7, finally, we present an dynamic evolving focus block tensor decomposition framework to deal with user focus shift challenge
- In Chapter 8, I conclude this dissertation.

Chapter 2

BACKGROUND AND RELATED WORKS

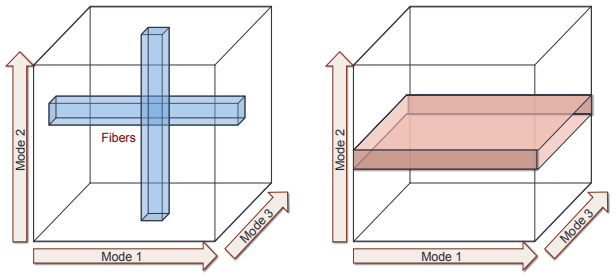
2.1 Tensor and Tensor Decomposition

In this section, the relevant background and notations are presented. Tensors are generalizations of matrices: while a matrix is essentially a 2-mode array, a tensor is an array of possibly larger number of modes. Intuitively, the tensor model maps a relational 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 [54] and the CANDECOMP/PARAFAC (or CP) [19] decompositions. Intuitively, both decompositions generalize singular value matrix decomposition (SVD) to tensor. CP decomposition, for example, decomposes the input tensor into a sum of component rank-one tensors.

2.1.1 Tensor Representations

A tensor is a multidimensional 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, an $M \times N \times K$ tensor of 3rd order has three modes: M columns (mode 1), N 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 [32] as shown in 2.1.



(a) Fiber (b) Slice

Figure 2.1: (A) a Fiber Is Defined by Fixing Every Index but One (B) a Slice Is Defined by Fixing All but Two Indices

For tensor operations, there is an important operation: Matricization, which transform a Tensor into a matrix by unfolding or flattening tensor along one mode. For instance, a $3\times4\times5$ tensor can be flattened as a 3×30 matrix, 4×15 matrix or 5×12 matrix. Generally, the mode-n matricization of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ is denoted by $\mathbf{X_{(n)}}$ and arranges the mode-n fibers to be the columns of the resulting matrix where $\mathbf{X_{(n)}} \in \mathbb{R}^{I_n\times(I_1\times\ldots\times I_{n-1}\times I_{n+1}\times\ldots\times I_N)}$.

Since matrices can be multiplied with other matrices or vectors, tensors can also be multiplied with other tensors such as matrices or vectors. For example, given an $M\times N\times K$ tensor and a $S\times N$ matrix, $\mathbf{A}$,

$$\boldsymbol{\mathcal{X}}' = \boldsymbol{\mathcal{X}}' \times_2 \mathbf{A}, \tag{2.1}$$

is an $M\times S\times K$ tensor where we can think as folding the result of $\mathbf{X}_{(2)}\times\mathbf{A}^T$ to become the result tensor $\mathcal{X}'$.

2.1.2 Tensor Decomposition

The two most popular tensor decomposition algorithms are the Tucker[54] and the CP[19] decompositions. Intuitively, both generalize singular value matrix decomposition (SVD) to tensors.

CP Decomposition The PARAFAC decomposition can be seen as a generalization of matrix factorizations in higher dimensions which are also referred to modes in tensor literatures [19]. PARAFAC decomposition is also known as CANDECOMP/PARAFAC (CP) decomposition. As shown in Figure 2.2, given a tensor $\boldsymbol{\mathcal{X}}$, CP factorizes the tensor into r component matrices (where r 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

$$\boldsymbol{\mathcal{X}} \approx \tilde{\boldsymbol{\mathcal{X}}} = recombine[\mathbf{A}, \mathbf{B}, \mathbf{C}] \equiv \sum_{r=1}^{R} a_r \circ b_r \circ c_r$$

where $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$ and $c_r \in \mathbb{R}^K$. The factor matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are the combinations of the rank-one component vectors into matrices; e.g., $\mathbf{A} = [\ a_1\ a_2 \cdots\ a_R\ ]$. This is visualized in Figure 2.2. As discussed in [32], 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 through 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 to the modes of the input tensor. Note that due to the approximate nature of tensor decomposition operation, given a decomposition $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ of $\boldsymbol{\mathcal{X}}$, the tensor $\tilde{\boldsymbol{\mathcal{X}}}$ that one would obtain by re-composing the tensor by combining 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:

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

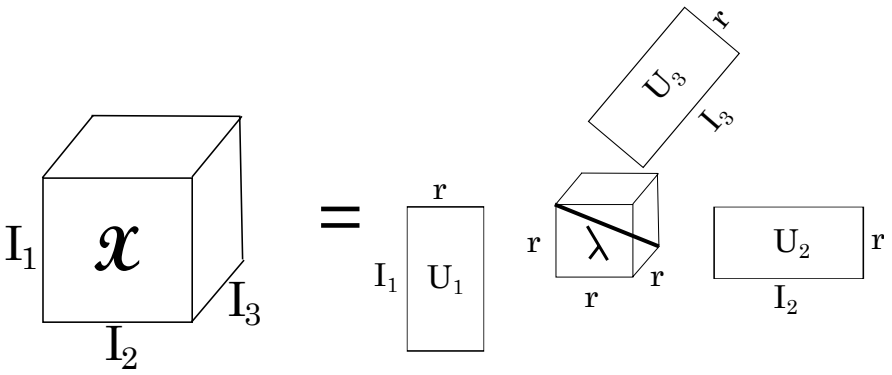


Figure 2.2: Cp-decomposition of a 3-mode Tensor Results in a Diagonal Core and Three Factors

2.2 Block-based Tensor Decomposition

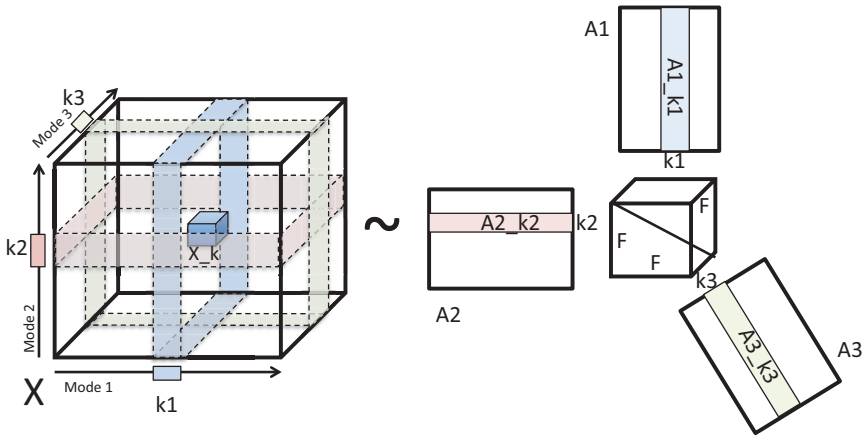


Figure 2.3: Each Sub-tensor (or Block) Can Be Described in Terms of the Corresponding Sub-factors

As the relevant data sets get large, existing in-memory schemes for tensor decomposition become increasingly ineffective and block-based solutions where some (possibly intermediate) data may be materialized on disks (instead of main memory) or other servers contributing to the decomposition process are necessitated. Several implementations of tensor decomposition operations on disk-resident data sets have been proposed. GridPARAFAC [39], 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.

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 $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ 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, R, for the tensor $\boldsymbol{\mathcal{X}}$. Let us further assume that each sub-tensor in $\mathfrak{X}$ has already been decomposed with target rank R 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

$$\boldsymbol{\mathcal{X}}_{\vec{k}} \approx \boldsymbol{I} \times_1 \boldsymbol{U}^{(1)}_{\vec{k}} \times_2 \boldsymbol{U}^{(2)}_{\vec{k}} \cdots \times_N \boldsymbol{U}^{(N)}_{\vec{k}}, \tag{2.2}$$

where $\boldsymbol{I}$ is the N-mode $R \times R \times \ldots \times R$ identity tensor, where the diagonal entries are all 1s and the rest are all 0s.

Given these, [39] presents an iterative improvement algorithm for composing these initial sub-factors into the full R-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:

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

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

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

terms of these sub-factors (Figure 2.3):

$$\boldsymbol{\mathcal{X}}_{\vec{k}} \approx \boldsymbol{I} \times_1 \boldsymbol{A}^{(1)}_{(k_1)} \times_2 \boldsymbol{A}^{(2)}_{(k_2)} \cdots \times_N \boldsymbol{A}^{(N)}_{(k_N)} \tag{2.3}$$

Moreover [39] shows that the current estimate of the sub-factor $\boldsymbol{A}^{(i)}_{(k_i)}$ can be revised using the update rule (for more details on the update rules please see [39]):

$$\boldsymbol{A}^{(i)}_{(k_i)} \longleftarrow \quad \boldsymbol{T}^{(i)}_{(k_i)} \left(\boldsymbol{S}^{(i)}_{(k_i)}\right)^{-1} \tag{2.4}$$

where

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

such that, 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
- $\boldsymbol{Q}_{\vec{l}} = \circledast_{h=1}^{N} (\boldsymbol{A}^{(h)T}_{(l_h)} \boldsymbol{A}^{(h)}_{(l_h)})$.

Above, $\circledast$ denotes the Hadamart product and $\oslash$ denotes the element-wise division operation. All these steps are included in Algorithm 1

Figure 2.4 shows an example of block-based tensor decomposition: Given input tensor $\boldsymbol{\mathcal{X}}$ is partitioned in to two sub-tensor, $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$. In the first stage, each sub-tensor is decomposed by CP and obtain the partial decomposed factors. The second stage combine those partial decomposed factors using iterative updates to derive decomposed factors of tensor $\boldsymbol{\mathcal{X}}$.

2.3 Two Phase Block-Centric CP Decomposition

Extended from block-based tensor decomposition, [34] proposed a two-phase block-centric CP decomposition shown in Algorithm 2. The difference between block-based and block-centric tensor decomposition is in Phase 2 where block-based CP decomposition will update

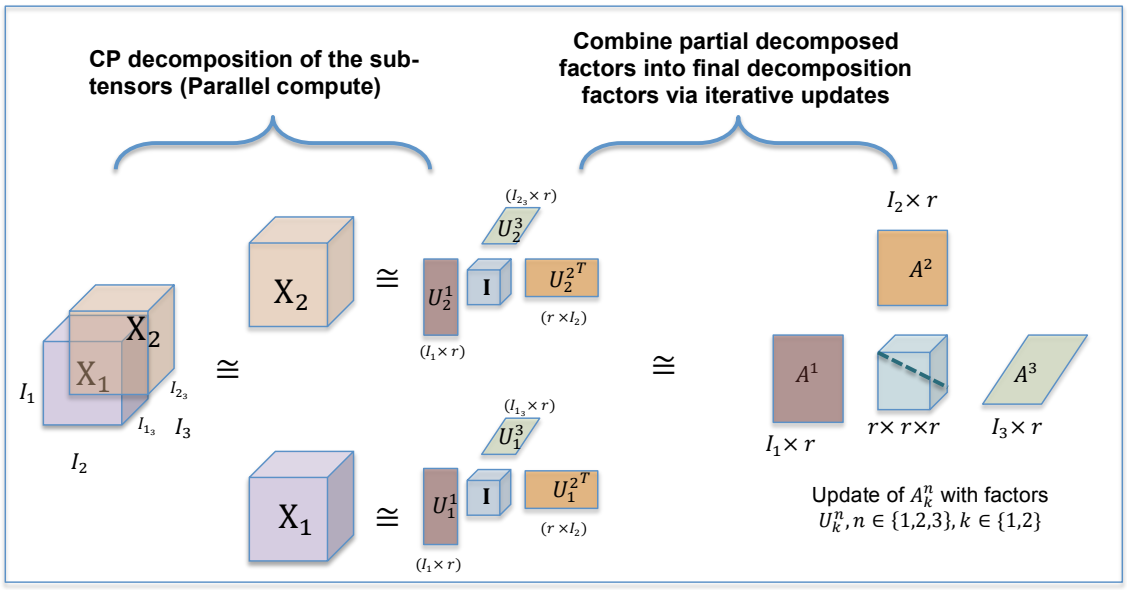


Figure 2.4: Illustration of Blocked-based Tensor Decomposition in Two Stages

the factors by iterating the mode index but block-centric CP decomposition will update the factors by iterating the block index. The block-centric updating order makes it more flexible to iterate the update unit and can reuse some existing part to efficiently compute the factor matrices

2.4 Personalized PageRank

As above stated, tensor can be partitioned into many sub-tensors,each sub-tensor have neighbors and connections with neighbors. This can form a network: each sub-tensor is a node, edge may be regarded as the connection between two sub-tensors. Therefore, we would like to capture the potential network topology to improve the block-based tensor decomposition. Therefore, we need a method to measure the significance of the node to analyze the connections among nodes.

The well-known PageRank algorithm [11] associates an importance score to each node relying on random walks: Let us consider a weighted, directed graph $G(V, E)$, where the

weight of the edge $e_j \in E$ is denoted as $w_j (\geq 0)$ and $\sum_{e_j \in outedge(v_i)} w_j = 1.0$. The PageRank score of nodes V is the stationary distribution of a random walk on G, denoted with a vector $\vec{p}$:

$$\vec{p} = (1 - \beta)\mathbf{T}_G \times \vec{p} + \beta\vec{v}, \tag{2.5}$$

wheore $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\vec{v}$ is a so-called *teleportation* vector, where all entries are $\frac{1}{\|V\|}$. An early attempt to contextualize the PageRank scores was the *topic sensitive PageRank* [22] approach which adjusts the PageRank scores of the nodes by assigning the *teleportation* probabilities in vector $\vec{j}$ in a way that reflects the graph nodes' degrees of match to a given search topic. In many applications, however, the context relevant to the recommendation is defined not through an explicit query, but a subset of the nodes (often referred to as the "*seed nodes*") in the graph. Personalized PageRank (PPR) [6, 14], for example, takes into account a user's interest by modifying the teleportation vector taking into account a given set of *important* nodes which are the target of the random jumps: given a set of nodes $S \subseteq V$, instead of jumping to a random node in V with probability β, the random walk jumps to one of the nodes in the seed set, S, given by the user. More specifically, if we denote the *Personalized PageRank* (PPR) scores of the nodes in V with a vector $\vec{\pi}$, then

$$\vec{\pi} = (1 - \beta)\mathbf{T}_G \times \vec{\pi} + \beta\vec{s}, \tag{2.6}$$

where $\vec{s}$ 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.

2.5 Incremental Tensor Analysis

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 [23, 35, 37, 44, 55]. [38] presented a pioneering work on up-

dating a tensor with PARAFAC decomposition, and applied it to MIMO radar application. [45] proposed tensor update algorithms: Dynamic Tensor Analysis (DTA), Streaming Tensor Analysis (STA), and Window-based Tensor Analysis (WTA). The idea behind DTA is to maintain some statistics that can easily be updated, therefore DTA obtains an update factor matrix by extracting leading eigenvectors of incrementally maintained covariance matrix in each mode and WTA combines the idea of sliding windows with DTA. STA is a fast algorithm of an approximate DTA by utilizing the tracking algorithm of incremental matrix decomposition. The idea behind the tracking algorithm is that for most of the cases, decomposition of tensor for each update can be very expensive especially when the change of covariance matrix is small, it is not worth re-decompose the tensor. SPIRIT is one of the fastest, such algorithms considers row insertions and deletions on matrices. Given a new row vector, for example, it first finds its projection y on the space defined by the current factor $\boldsymbol{U}$, by projecting x onto $\boldsymbol{U}$. Given this projection, the 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", 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 to capture the new row vector. The "forgetting factor" is used to control the speed with which $\boldsymbol{U}$ is revised: if the factor is high, $\boldsymbol{U}$ is revised quickly to match the new data; if, in contrast, the factor is low, $\boldsymbol{U}$ is preserved even if it does not account well for the new vector.

Algorithm 3 introduces how SPIRIT is generalized to tensor that maintain the factor matrices of an updated tensor. Note that, this process run on the matricization of the tensor on each mode. For each column vector of the matricization of tensor, it conducts the tracking process. Therefore, for large tensors, processing on each vectors of each mode matricization is still expensive. [45] further reduce the time complexity by selecting only a subset of the vectors in matricization matrix. For example, they can sample vectors with high norms

because these potentially give higher impact to the projection matrix.

In this section, the relevant background and notations are presented. Tensors are generalizations of matrices: while a matrix is essentially a 2-mode array, a tensor is an array of possibly larger number of modes. Intuitively, the tensor model maps a relational 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 [54] and the CANDECOMP/PARAFAC (or CP) [19] decompositions. Intuitively, both decompositions generalize singular value matrix decomposition (SVD) to tensor. CP decomposition, for example, decomposes the input tensor into a sum of component rank-one tensors.

More specifically, given a tensor $\boldsymbol{\mathcal{X}}$, the CP decomposition 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 three matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$, such that

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

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., $\mathbf{A} = [\ a_1\ a_2\ \cdots\ a_F\ ]$. This is visualized in Figure 2.5.

Many algorithms for decomposing tensors are based on an iterative process that tries to improve the approximation until a convergence condition is reached through 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 to the modes of the input tensor. Note that due to the approximate nature of tensor decomposition operation, given a decomposition $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ of $\boldsymbol{\mathcal{X}}$, the tensor $\tilde{\boldsymbol{\mathcal{X}}}$ that one would obtain by re-composing the tensor by combining 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

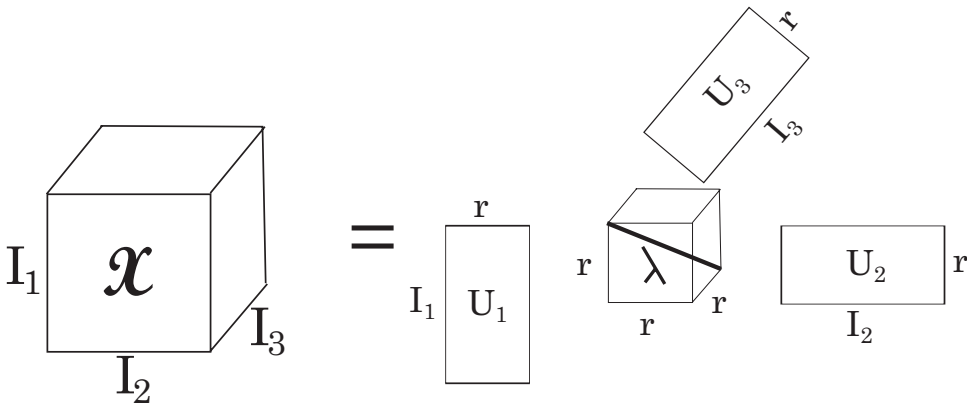


Figure 2.5: Illustration of CP Decomposition

measured by considering the Frobenius norm of the difference tensor:

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

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

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

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
   (a) for each mode $i = 1$ to N
      i. for each modal partition, $k_i = 1$ to K_i,
         A. update $\boldsymbol{A}^{(i)}_{(k_i)}$ 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 $\boldsymbol{P}_{[*,\ldots,*,k_i,*,\ldots,*]}$ using $\boldsymbol{U}^{(i)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ and $\boldsymbol{A}^{(i)}_{(k_i)}$
            - compute $\boldsymbol{S}^{(i)}_{(k_i)}$ using the above
            - update $\boldsymbol{A}^{(i)}_{(k_i)}$ using the above
            - for each $\vec{k} = [*, *, \ldots, k_i, \ldots, *, *]$
               – update $\boldsymbol{P}_{\vec{k}}$ and $\boldsymbol{Q}_{\vec{k}}$ using
               – $\boldsymbol{U}^{(i)}_{\vec{k}}$ and $\boldsymbol{A}^{(i)}_{(k_i)}$

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

Algorithm 2 Two-Phase Block-Centric CP Decomp. [34]

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 $\mathring{\boldsymbol{\mathcal{X}}}$

Algorithm 3 SPIRIT-based tensor factor tracking algorithm[45]

Input: incremental tensors, $\mathcal{X}$, matricization matrix, $\mathbf{X_{(n)}} \in \mathbb{R}^{I_n \times (I_1 \times ... \times I_{n-1} \times I_{n+1} \times ... \times I_M)}$, the number of modes, M, old decomposition factor $\boldsymbol{\mathcal{U}}^d$ where $d = 1, ..., M$, energy matrix $\mathcal{S}$, rank r and forgetting factor λ

Output: sub-tensor decomposition factor $\boldsymbol{\mathcal{U}}$ and energy matrix $\mathcal{S}$

for $n = 1 \rightarrow M$ **do**
  $\mathbf{X_{(n)}} = matricization_n(\mathcal{X})$
  for *each column vector* x *in* $\mathbf{X_{(n)}}$ **do**
    Initialize $\hat{x}_1 := x$
    for $i = 1 \rightarrow r$ **do**
      $y_i := u_m(i)^T \hat{x}_i$
      $s_i \leftarrow \lambda s_i + y_i^2$
      $e_i := x_i - y_i u_m(i)$
      $u_m(i) \leftarrow u_m(i) + \frac{1}{s_i} y_i e_i$
      $\hat{x}_{i+1} := \hat{x}_i - y_i u_m(i)$
    end
  end
end

Chapter 3

SIG: SUB-TENSOR IMPACT GRAPHS

3.1 Introduction

Tensor decomposition process generalizes matrix decomposition to high-dimensional arrays and the resulting factor matrices and core tensors which 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 [19] and Tucker [54]) are proven to be effective in multi-aspect data analysis and clustering. For instance, [43] 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. [56] 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. [48], on the other hand, has shown that Tucker decomposition can be used for subspace clustering which simultaneously conducts dimensionality reduction and membership representation.

3.1.1 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 [24] focuses

on sparse tensors and presents a scalable tensor decomposition suite of methods for Tucker and PARAFAC decompositions on the MapReduce framework. TensorDB [28, 29] 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 [39], 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. Section 2.2 and 2.3 provide an overview of the block-based tensor decomposition process.

The block-based tensor decomposition process is outlined in pseudcode in Algorithm 1. Figure 2.4 provides a visual example of this process: The given input tensor $\boldsymbol{\mathcal{X}}$ is partitioned into two sub-tensors, $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$. In the first stage, each sub-tensor is decomposed by CP, thus obtaining partial factors. The second stage combines these partial decomposed factors using iterative updates to derive the final factors (and the corresponding core) for tensor $\boldsymbol{\mathcal{X}}$.

3.1.2 Contributions of this Chapter: Sub-Tensor Impact Graphs

While block-based tensor decomposition techniques [39, 28] 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*.

Key Observations

As described above, block-based tensor decomposition techniques (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 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 where any inaccuracies in individual sub-tensor decompositions can propagate (through the update rules introduced in Section 2.2) 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, I introduced the notion of *sub-tensor impact graphs (SIGs)*, 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.

Challenge #1: Decomposition in the Presence of Data Evolution

Firstly, we rely on sub-tensor impact graphs (SIGs) to tackle performance challenges that dynamic data pose in tensor analytics: incremental tensor decomposition. Re-computation 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 Section 3.3, I present several optimization strategies based on sub-tensor impact graphs to prune unnecessary computation in the presence of incremental updates on the data.

Challenge #2: Personalization of the Decomposition on User Focus

Besides, I present alternative ways to account for the impact of the accuracy of one region (user focus) 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, I present an optimization strategy to boost the final decomposition accuracies for partitions of user's focus.

3.2 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, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, be partitioned into a grid, $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$, of sub-tensors, such that

- K_i indicates the number of partitions along mode-i,
- the size of the j^{th} 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 $\prod_{i=1}^{N} K_i$.

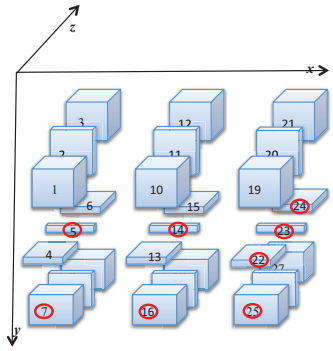


Figure 3.1: A Sample 3-mode Tensor, Partitioned into 27 Heterogeneous Sub-tensors

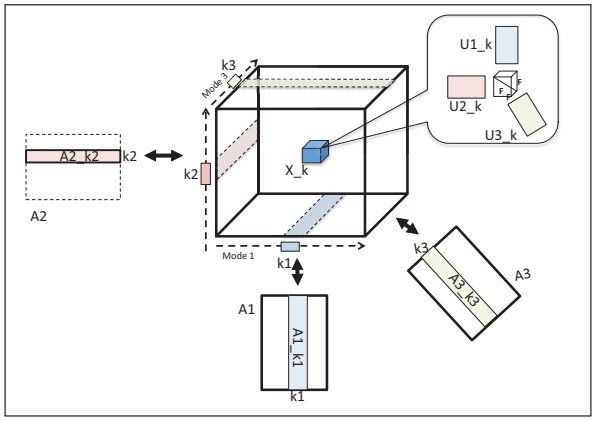


Figure 3.2: The Block-based Update Rule Maintains $\boldsymbol{A}^{(i)}_{(k_i)}$ Incrementally by Using the Current Estimates for $\boldsymbol{A}^{(j)}_{(k_j)}$ and the Decompositions In $\mathfrak{U}^{(j)}$

Example 3.2.1 *Figure 3.1 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 3 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.*

3.2.1 Accuracy Dependency among Sub-Tensors

In Section 2.2, 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 are not critical for our discussion (and is beyond the scope of this chapter), it is **important** to note that, as visualized in Figure 3.2, each $\boldsymbol{A}^{(i)}_{(k_i)}$ is maintained incrementally by using, for all $1 \leq j \leq N$, the current estimates for $\boldsymbol{A}^{(j)}_{(k_j)}$ 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. Moreover, and most importantly for the present discussion, this update rule for $\boldsymbol{A}^{(i)}_{(k_i)}$ supports the following observation: Given

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

the final accuracy for the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, $\vec{k} = [k_1, \ldots, k_i, \ldots, k_N]$, depends on the accuracies of sub-factors $\boldsymbol{A}^{(i)}_{(k_i)}$. 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 $\boldsymbol{A}^{(i)}_{(k_i)}$, we need to 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 $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$), and
- $\boldsymbol{P}_{[*,\ldots,*,k_i,*,\ldots,*]}$, which in turn uses $\boldsymbol{U}^{(h)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ for $1 \leq h \leq N$ (i.e., all factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$).

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

In other words, for each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, there is a set, $direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{k}}) \subseteq \mathfrak{X}$, of sub-tensors that consists of those sub-tensors whose initial decomposition accuracies directly impact the final decomposition accuracy of $\boldsymbol{\mathcal{X}}_{\vec{k}}$. Moreover, as visualized in Figure 3.3,

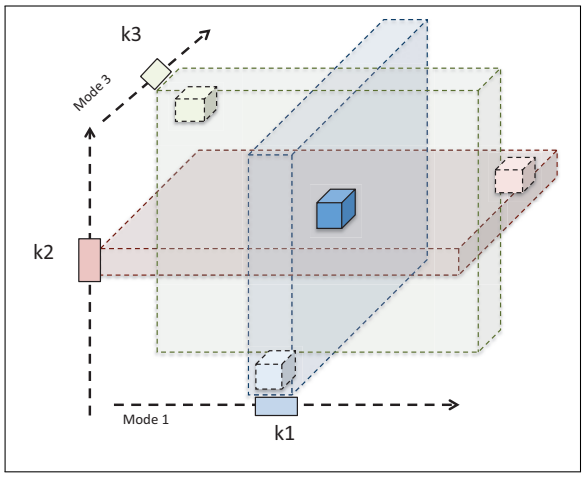


Figure 3.3: The Sub-tensors Whose Initial Decomposition Accuracies Directly Impact given Sub-tensors Are Aligned with That Sub-tensor along with the Different Modes of the Tensor.

$direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{k}})$ consists of those sub-tensors that are aligned (i.e., share the same slices) with $\boldsymbol{\mathcal{X}}_{\vec{k}}$, along the different modes of the tensor.

3.2.2 Sub-Tensor Impact Graphs (SIGs)

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

Definition 3.2.1 *Let an N-mode tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, be partitioned into a grid, $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$, of sub-tensors. The corresponding* sub-tensor impact graph (SIG) *is a directed, weighted graph, $G(V, E, w())$, where*

- *for each $\boldsymbol{\mathcal{X}}_{\vec{k}} \in \boldsymbol{\mathcal{X}}$, 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, and*
- *$w()$ is an edge weight function, such that $w(v_{\vec{l}} \rightarrow v_{\vec{k}})$ quantifies the direct accuracy impact of decomposition accuracy of $\boldsymbol{\mathcal{X}}_{\vec{l}}$ on $\boldsymbol{\mathcal{X}}_{\vec{k}}$.*

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.

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 $w_{uni}(e) = 1$.

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 Figure 3.1, 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 $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$, defined as below, may need to be considered to account for impact propagation:

Definition 3.2.2 (Surface of Interaction) *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*
- $\vec{l} = [k_{l_1}, k_{l_2}, \ldots, k_{l_N}]$.

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

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

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 $(v_{\vec{j}} \rightarrow v_{\vec{l}}) \in E$ be an edge in the graph. The weight of this edge from $v_{\vec{j}}$ to $v_{\vec{l}}$ should reflect the area of the* surface of interaction *between the sub-tensors $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$.*

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 $(v_{\vec{j}} \rightarrow v_{\vec{l}}) \in E$ as follows:

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

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 $\boldsymbol{\mathcal{X}}$, 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 3.2.3 (Value Alignment) *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 the vector $\vec{\boldsymbol{c}}_{\vec{j}}(A)$ *is constructed from the sub-tensor* $\boldsymbol{\mathcal{X}}_{\vec{j}}$ *as follows*[1]*:*

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

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

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

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

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

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

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 $(v_{\vec{j}} \rightarrow v_{\vec{l}}) \in E$ in the sub-tensor impact graph as follows:

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

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

Alt. #4: Combined Edge Weights

The suroface 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:

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

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

3.2.3 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 m^{th} 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 [6]) 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 Sections 3 through 3.2.2 for various applications).

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

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

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. Intuitively, $\vec{p}$ 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 $\vec{p}$ by solving the above equation[2]. 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.

3.3 Optimization Strategies

To deal with the challenges in the presence of data evolution, user focus and potential user focus shifting, I proposed several strategies to optimize block-based tensor decomposition relies on SIG and impact score. Here the optimization strategy focused on phase-2 iterative refinement process of block-based tensor decomposition described in Sectin 3.1.1. Based on block-based tensor decomposition, we have several key observations that support the optimization:

3.3.1 Observation-1: Sub-Tensor Rank Flexibility

Remember from Section 2.2, 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 Figure 2.3) each $\boldsymbol{A}^{(i)}_{(k_i)}$ is maintained incrementally by using, for all $1 \leq j \leq N$, the current estimates for $\boldsymbol{A}^{(j)}_{(k_j)}$ 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 $\boldsymbol{A}^{(i)}_{(k_i)}$ 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.

[2]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.

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, $\boldsymbol{A}^{(i)}$, for $\boldsymbol{\mathcal{X}}$.

3.3.2 Observation-2: Accuracy Dependency among Sub-tensors

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

From the above formular, it is easy to see that the final accuracy for the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, $\vec{k} = [k_1, \ldots, k_i, \ldots, k_N]$, depends on the accuracies of sub-factors $\boldsymbol{A}^{(i)}_{(k_i)}$. 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 $\boldsymbol{A}^{(i)}_{(k_i)}$, we need to 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 $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$), and
- $\boldsymbol{P}_{[*,\ldots,*,k_i,*,\ldots,*]}$, which in turn uses $\boldsymbol{U}^{(h)}_{[*,\ldots,*,k_i,*,\ldots,*]}$ for $1 \leq h \leq N$ (i.e., all factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,k_i,*,\ldots,*]}$).

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

In other words, for each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, there is a set, $direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{k}}) \subseteq \boldsymbol{\mathfrak{X}}$, of sub-tensors that consists of those sub-tensors whose initial decomposition accuracies directly impact the final decomposition accuracy of $\boldsymbol{\mathcal{X}}_{\vec{k}}$. Moreover, as visualized in Figure 3.3, $direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{k}})$ consists of those sub-tensors that are aligned (i.e., share the same slices) with $\boldsymbol{\mathcal{X}}_{\vec{k}}$, along the different modes of the tensor.

3.3.3 Optimization-1: Decomposition Rank Assignment based on Impact Score

Based on sub-tensor rank flexibility property described in Section 3.3.1, decomposition rank of sub-tensors could be different. 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 proposed in this chapter, we constructs a *sub-tensor impact graph*, G, that accounts for the propagation of inaccuracies along the tensor during a block-based decomposition process. The block-based CP decomposition then leverages this graph to account for the impact of the initial decomposition inaccuracy of one sub-tensor on the final decomposition accuracy of $\boldsymbol{\mathcal{X}}_P$; i.e., the cells of $\boldsymbol{\mathcal{X}}$ collectively covered by the user's declaration of interest (i.e., $\mathcal{K}_P$).

Intuitively, the initial decomposition rank, $F_{\vec{k}}$, of sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, will need to reflect the impact of the initial decomposition of the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$ on the final decomposition of the high-priority sub-tensors, $\boldsymbol{\mathcal{X}}_{\vec{\kappa}}$, $\vec{\kappa} \in \mathcal{K}_P$. This implies that, when picking the decomposition ranks, $F_{\vec{k}}$, 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 Section 3; more specifically, we compute the initial decomposition rank, $F_{\vec{k}}$ of $\boldsymbol{\mathcal{X}}_{\vec{k}}$ as

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

where $\vec{s}[k]$ denotes the sub-tensor impact score of the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$ 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)[3] based on how far they are from the seed set in the sub-tensor impact graph, G.

[3]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.

3.3.4 Optimization-2: Ignore Low Impact Sub-tensor in the Refinement

Based on Observation-2, those sub-tensors that have direct refinement relationships with the updated sub-tensors are critical to the refinement process. 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 of block-based tensor decomposition described in Section 3.1.1. Sub-tensor impact graph is proposed to account for the inaccuracy propagation and impact score is used to present the sub-tensor decomposition impact on the others' decomposition.

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.

3.3.5 Optimization-3: Assign Probability of Refinement based on Impact Score

While optimization-2 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.

3.3.6 Summary

To deal with tensor decomposition challenges, block-based tensor decomposition partition tensor into sub-tensors and execute in two phases: phase-1, sub-tensors are decomposed

in parallel; phase-2, sub-tensors' factors are combined through iterative refinement process to compose final tensor decomposition factor. Sub-tensor impact graph is proposed to account for the inaccuracy propagation and impact score is used to present the sub-tensor decomposition impact on the others' decomposition.

Based on the key observations, I proposed three optimization strategies to optimize phase-2 which can be used to optimize different applications of tensor decomposition and will be further discussed in Section 5, Section 6 and Section 7

Chapter 4

RPR: ROBUST PERSONALIZED PAGERANK

4.1 Introduction

How a given pair of nodes in a social network are related to each other reflects the underlying network topology. Recommendation systems often use the analysis of the structure of a given data and/or social graph, relative to the user's current context, to generate rankings and recommendations [18].

Significance of a node in a graph needs to reflect both the topology of the graph and the application semantics and measures: The *betweenness* measure [57] for example aims to quantify whether deleting the node would disconnect or disrupt the graph. The *centrality/cohesion* [10] measures quantify how close to a clique the given node and its neighbors are. Other *authority*, *prestige*, and *prominence* measures [6, 11, 10] measure the significance of the node in the graph through eigen-analysis or random walks.

For example, the well-known PageRank algorithm [11] associates an importance score to each node relying on random walks: Let us consider a weighted, directed graph $G(V, E)$, where the weight of the edge $e_j \in E$ is denoted as $w_j (\geq 0)$ and $\sum_{e_j \in outedge(v_i)} w_j = 1.0$. The PageRank score of nodes V is the stationary distribution of a random walk on G, denoted with a vector $\vec{p}$:

$$\vec{p} = (1 - \beta)\mathbf{T}_G \times \vec{p} + \beta\vec{v}, \tag{4.1}$$

where $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\vec{v}$ is a so-called *teleportation* vector, where all entries are $\frac{1}{\|V\|}$.

An early attempt to contextualize the PageRank scores was the *topic sensitive PageRank* [22] approach which adjusts the PageRank scores of the nodes by assigning the *telepor-*

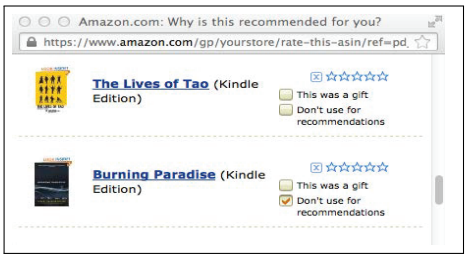


Figure 4.1: An Example Interface Enabling the User to Explicitly Eliminate Outliers in Generating Recommendations; Such Explicit Corrections Is Not Feasible in All Applications of Social Networks

tation probabilities in vector $\vec{j}$ in a way that reflects the graph nodes' degrees of match to a given search topic. In many applications, however, the context relevant to the recommendation is defined not through an explicit query, but a subset of the nodes (often referred to as the "*seed nodes*") in the graph. Personalized PageRank (PPR) [6, 14], for example, takes into account a user's interest by modifying the teleportation vector taking into account a given set of *important* nodes which are the target of the random jumps: given a set of nodes $S \subseteq V$, instead of jumping to a random node in V with probability β, the random walk jumps to one of the nodes in the seed set, S, given by the user. More specifically, if we denote the *Personalized PageRank* (PPR) scores of the nodes in V with a vector $\vec{\pi}$, then

$$\vec{\pi} = (1 - \beta)\mathbf{T}_G \times \vec{\pi} + \beta\vec{s}, \tag{4.2}$$

where $\vec{s}$ 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.

4.1.1 Problem - Noisy Seed Sets

The above formulation of PPR *assumes that all seeds are equally important* in characterizing the user's interest. This, however, may not always be the case, since in practice, the user feedback is often incomplete and noisy [12] (Figure 4.1). Unfortunately unless (a) each *individual* seed node is a good representative for the entire seed set, (b) the user/system was successful in including all seed nodes relevant for defining the current user context, and

Rank Statistics of 9 Noisy Seeds (out of 49 Seeds in 2500 Movies)		
Best Rank	*Avg. Rank*	*Worst Rank*
18	42.9	52

Table 4.1: Bias and Lack of Seed Differentiation in PPR Scores: Most of the 49 Seeds (out of 2500 Movies) Have Very High PPR Ranks, Even If They Are Outliers in the Seed Set

(c) most importantly, the user/system did not include any outlier nodes in the seed set, the resulting rankings might be biased and might contain undesirable artifacts, such as movies with low ratings, having high PPR rankings. Consider the following example:

Example 4.1.1 *(Impact of the Noise in the Seed Set) Table 4.1 shows the PPR scores and ranks of the noisy seed nodes in a movie graph (see Section 5.3 for more details about this data set) for a sample user. In this example, we construct an* imperfect seed set *as follows: we select a random user and include in the seed set 40 movies rated as "like" and 9 movies rated as "dislike" by this user (out of 81 movies rated like and 25 movies rated dislike; by this user).*

Table 4.1 studies the relationship between the original user rating and the PPR ranking. As we see here, while as expected the average PPR rankings of the 9 noisy seed movies is poor among the rankings of the seed movies (~ 43 *out of* 52*), it is also true that all the seed movies (including the 9 dislike-rated outlier movies) rank highly (i.e., better than* 53 *out of* 2500*).*

Note that, while we often do not care about the ranks of the seed movies, high PPR-ranking of dislike-rated seeds implies that those movies neighboring these noisy seeds are also likely to be ranked highly.

4.1.2 Our Contributions: Robust Personalized PageRank (RPR)

Intuitively, a *noisy seed* is a seed node (provided by the user or selected by the system) which does not properly reflect the user's focus in that it does to fit in the context defined by the seed set as a whole. The above example illustrates that *a poor seed may overestimate the rankings of its (equally poor) neighbors in the final ranking.* This is primarily due to *(a) Teleportation-bias:* Firstly, the teleportation-vector based seeding algorithms jump on the seed set, S, relatively often (the common transportation probability, β, is 0.15) and the size of the seed set is often much smaller relative to the size of the data graph: as a result, the PPR value of the least significant seed node will be at least $\frac{\beta}{|S|}$, which is likely to be much higher than the PPR of non-seed nodes in the graph. *(b) Need for seed differentiation:* Secondly, as we commented above, the negative impact of the teleportation-bias can be alleviated if the teleportation rates to the seeds can be *differentiated* identify which seeds are truly better than others as this information is not available *a priori*[1]:

In this work, we propose *techniques to eliminate teleportation-bias and/or provide seed differentiation*:

- First and foremost, we discuss how the node rankings can be negatively affected by possible incompleteness and/or imperfection in the seeds set, and we experimentally establish that the conventional PPR metrics might not properly differentiate seed nodes in a graph.

- Secondly, we propose alternative *Robust personalized PageRank* (RPR) strategies, (a) which are insensitive to noise in the set of seed nodes (and thus differentiate seeds well) and (b) in which the rankings are not overly biased towards the seed nodes (Table 4.2).

[1]Otherwise the seed set would have been constructed differently

Rank Statistics of 9 Noisy Seeds (out of 49 Seeds in 2500 Movies)		
Best Rank	*Avg. Rank*	*Worst Rank*
94	1109.8	1568

Table 4.2: Seed Differentiation and Bias Elimination Through *Robust Personalized Pagerank* (RPR) Scores: In the Same Situation as In Table 4.1, Average Rating of the Noisy Seeds Is Greater than 1000 out of 2500 Movies

4.2 Robust Personalized PageRank

Addressing the problem of noisy seeds through non-uniform teleportation to the seed nodes requires a mechanism to distinguish among the nodes in the seed set, S.

4.2.1 PPR-G: PPR with Global Seed Ranking

A *first (and as we see later, mostly ineffective) attempt* to differentiate among the seeds might be to consider the global properties of the nodes in the seed set. One relatively straightforward way to achieve this is to first measure the significance of the individual seed nodes in the overall graph, G, (using for example PageRank) and, then, modulate the teleportation rates onto the seed nodes based on the relative significance values of the seeds. In other words, we would compute the *PPR scores with global seed ranking* (also referred to as PPR-G scores) as follows: Given a graph, $G(V, E)$, and a seed node set S, let $\vec{p}$ be the PageRank scores computed by solving Equation 4.1. Then, the PPR-G scores, $\vec{g}$, are obtained by solving

$$\vec{g} = (1 - \beta)\mathbf{T}_G \, \vec{g} + \beta \vec{s_2},$$

where $\vec{s_2}$ is a re-seeding vector, such that for each $v_i \notin S$, $\vec{s_2}[i] = 0$, and for each $v_i \in S$, $\vec{s_2}[i] = \frac{\vec{p}[i]}{\sum_{v_j \in S} \vec{p}[j]}$.

As we later see in Section 5.3, however, in many cases, simply modifying the teleportation rates of the seed nodes based on the global significances of the nodes in the seed set does not properly eliminate seed-bias.

4.2.2 RPR-1: Teleportation-Discounted PPR

In the PPR formulation, the seed and non-seed nodes have different contributors to their final scores. *For the non-seed nodes*, the only contributor to their PPR scores is the number of times they are visited during the regular random-walk process. On the other hand, *for the seed nodes*, both (a) the number of times they are visited during the regular random-walk process and (b) the number of times they are selected as a teleportation destination for random-walk restart contribute to the PPR scores; we refer to these as the *random-walk contribution* (`rw-PPR`) and *teleportation contribution* (`t-PPR`), respectively. As described above, for non-seed nodes, the value of `t-PPR` score is 0.0.

Our first observation is that the `t-PPR` score of a seed node is exactly $\beta/|S|$ for each seed and, thus, a seed node will have at least $\beta/|S|$ overall PPR score, even if it is an outlier in the overall context defined by the seed set, S. Therefore, the first proposal to increase robustness of the the PPR scores against noise in the the seed set S is to discount the teleportation contributions from the PPR scores.

Teleportation-Discounting

Based on this observation, we define *teleportation-discounted PPR* scores (also referred to as RPR-1 scores) as follows: Given a graph, G, and a seed node set S, let $\vec{\pi}$ be the PPR scores as defined earlier:

- for each $v_i \notin S$, the corresponding RPR-1 score, $\vec{\rho_1}[i]$, is defined as $\vec{\rho_1}[i] = \frac{\vec{\pi}[i]}{1-\beta}$;

(a) A Is Under-accounted Relative to B and B If We Discount Teleportations

(b) A Is Not Under-accounted When Teleportation Scores Are Discounted

Figure 4.2: (A) Discounting Teleportation Scores Would Cause Under-accounting of A Relative to B and C. (B) If We Add Self-loops to Nodes, When A Is Selected as a Re-start Point, B Will Not Be Under-accounted Relative to Its Neighbors

- in contrast, for each $v_i \in S$, the corresponding RPR-1 score, $\vec{\rho_1}[i]$, is defined as $\vec{\rho_1}[i] = \frac{\vec{\pi}[i] - \frac{\beta}{|S|}}{1-\beta}$.

PPR-1 scores as defined above do not alter the relative ordering of the non-seed nodes; instead (as discussed above) they aim to allow us to discover the significance of the seed nodes themselves relative to the non-seed nodes within the overall context defined by the seed set, S.

Preventing Under-Accounting of Seeds

One potential problem with the teleportation-discounting is that this time the seed nodes may in fact be *under-accounted*: neighbors of a seed node may get higher amounts of random-walk traffic (i.e, `rw-PPR`) than the seed node itself, simply because once you teleport to a seed node, you need to visit one of its neighbors but not vice versa. In order to prevent this *under-accounting*, we modify the input graph G by inserting a self-loop to each node in the network (Figure 4.2). In the resulting graph G', every node is a neighbor of itself and thus a seed node, v, will not get a lesser amount of random-walk traffic than its neighbors when v is selected as a re-start point.

4.2.3 RPR-2: Seed-Set Maximal PPR

RPR-1 reduces the teleportation bias, but the seed nodes are still being teleported to with the same, undifferentiated rate.

Seed-Set Maximality Principle and RPR-2

An alternative to fixing the teleportation significance of the individual seed nodes *a priori*, without considering the context provided by the other seed nodes (as in the PPR-G scheme we discussed in Section 4.2.1), is to discover the teleportation significance of the individual seed nodes, relying on a novel *seed-set maximality* principle that would tie the teleportation rates of the seeds to their contributions to the overall personalized PageRank score of the seed set.

Principle 3 (Seed-Set Maximal PPR Scores) Let $G(V, E)$ be a graph and let $S \subseteq V$ be a set of seed nodes. Given an overall teleportation rate β, the re-seeding/re-start vector, $\vec{s}$, should be selected such that the overall PageRank scores of the nodes in S will be maximal. $\Diamond$

Intuitively, this principle requires that the seed set as a whole must score highly, but does not require that the individual seed nodes themselves have high scores. Based on this principle, the *seed-set maximal PPR scores* (also referred to as the RPR-2 scores) are computed as follows: Given a graph $G(V, E)$, a teleportation probability, β, and a seed set, S, the re-start vector $\vec{s}$ should be such that

$$\vec{\rho_2} = (1 - \beta)\mathbf{T}_G \, \vec{\rho_2} + \beta\vec{s},$$

$$\sum_{v_i \in V} \vec{s}[i] = 1, \forall_{v_i \in V} 0 \leq \vec{s}[i] \leq 1, \sum_{v_i \in V} \vec{\rho_2}[i] = 1, 0 \leq \vec{\rho_2}[i] \leq 1,$$

and the following term is *maximized*:

$$seed\_set\_significance = \sum_{v_i \in S} \vec{\rho_2}[i].$$

Naturally, one possible concern with this new formulation is that it might potentially increase the computation cost. We next show that RPR-2 scores can be cheaply computed.

Seed only Re-Starts for RPR-2

According to the above formulation of seed-set maximal PPR scores, the seed nodes in S are not necessarily the only targets for re-starts. However, we can show that, for any traversal that re-starts at a non-seed node, there is a traversal that starts only at the seed nodes, but has a higher seed node traversal rate.

Proof Sketch: Given a graph $G(V, E)$, let $\vec{s}$ be an optimal re-start vector, such that $\exists v_i \notin S$ such that $\vec{s}[i] > 0$. Now consider the traversals that start only from a $v_i \notin S$ and let $\vec{\alpha_i}$ be a vector describing the average portion of time the random walk spends on the graph nodes in V before the next teleportation. We can then define two quantities (that add up to 1.0):

$$seed\_ratio_i = \sum_{v_j \in S} \vec{\alpha_i}[j], \text{ and } non\_seed\_ratio_i = \sum_{v_j \notin S} \vec{\alpha_i}[j].$$

Note that, since the traversal starts at a non-seed node, we have $non\_seed\_ratio_i > 0$; moreover, we can split this term into two, $non\_seed\_ratio_{i,before}$ and $non\_seed\_ratio_{i,after}$; i.e., the amount of time spent on non-seed nodes before and after a seed node is met during the random-walk, respectively. Let us also define a vector, $\vec{f_i}$, where for a given $v_j \in S$, the value of $\vec{f_i}[j]$ is the likelihood of v_j being the first seed met during the random-walk starting at node v_i.

Now consider an alternative re-start vector $\vec{\sigma}$ such that (a) $\vec{\sigma}[i] = 0$, (b) $\forall_{v_j \notin S}$, if $j \neq i$, then $\vec{\sigma}[j] = \vec{s}[j]$, and (c) $\forall_{v_j \in S}$, $\vec{\sigma}[j] = \vec{s}[j] + \vec{s}[i]\ \vec{f_i}[j]$. It is easy to see that the random-walks resulting when using the restart vector $\vec{\sigma}$ are similar to the random-walks resulting when using $\vec{s}$, except that the value of $non\_seed\_ratio_{i,before}$ is equal to 0. This means some of this time will be spent on the seed nodes contradicting the initial premise that $\vec{s}$ was an optimal re-start vector, maximizing the total amount of time spent on seed nodes. ■ Based on the

above proof, we can reformulate the equation set for seed-set maximal PPR scores in a way that limits the transportation targets, as follows:

$$\vec{\rho_2} = (1-\beta)\mathbf{T}_G\ \vec{\rho_2} + \beta\vec{s},\ \sum_{v_i \in V} \vec{\rho_2}[i] = 1, \quad 0 \leq \vec{\rho_2}[i] \leq 1,$$

$$\sum_{v_i \in S} \vec{s}[i] = 1, \quad \forall_{v_i \in s} 0 \leq \vec{s}[i] \leq 1, \quad \forall_{v_i \notin s} \vec{s}[i] = 0,$$

and $\quad seed\_set\_significance = \sum_{v_i \in S} \vec{\rho_2}[i] \quad$ is *maximum.*

Constraining the Size of the Re-Start Set

We can further show that in practice the restart set, S_{crit}, is a singleton; i.e., with very high likelihood, $|S_{crit}| = 1$. **Proof Sketch:** Given a graph $G(V, E)$, let $\vec{s}$ be an optimal re-start vector, such that $\exists v_i, v_j \in S$ such that $\vec{s}[i] > 0$ and $\vec{s}[j] > 0$. Now consider all the traversals that start only from v_i and let $\vec{\alpha_i}$ be a vector describing the average portion of time the random walk that starts at v_i spends on the nodes in V before the next teleportation. Similarly, let $\vec{\alpha_j}$ be a vector describing the average portion of time a random walk that starts at v_j spends on the nodes in V before the next teleportation. Given $\vec{\alpha_i}$ and $\vec{\alpha_j}$, let us define two quantities,

$$seed\_ratio_i = \sum_{v_k \in S} \vec{\alpha_i}[k] \ \text{ and } \ seed\_ratio_j = \sum_{v_k \in S} \vec{\alpha_j}[k],$$

and let us assume that $seed\_ratio_i > seed\_ratio_j$ (a similar argument holds when $seed\_ratio_j > seed\_ratio_i$).

Now consider an alternative re-start vector $\vec{\sigma}$ such that (a) $\forall_{v_k \notin \{v_i, v_j\}}$, $\vec{\sigma}[k] = \vec{s}[k]$, (b) $\vec{\sigma}[j] = 0$, and (c) $\vec{\sigma}[i] = \vec{s}[i] + \vec{s}[j]$. It is easy to see that in the random-walks resulting when using the restart vector $\vec{\sigma}$ are similar to the random-walks resulting when using $\vec{s}$, except that all transportations to v_j are replaced with transportations to v_i (with the higher *seed_ratio* value among the two); thus, overall, more time will be spent on seed nodes when using $\vec{\sigma}$ instead of $\vec{s}$. It follows that when $seed\_ratio_i > seed\_ratio_j$, an optimal re-start vector cannot

contain both v_i and v_j, contradicting the initial premise that $\vec{s}$ is an optimal re-start vector, such that both $\vec{s}[i] > 0$ and $\vec{s}[j] > 0$. ■ In other words, since in practice it is highly unlikely that *seed_ratio* values will be equivalent for different seed nodes, the subset S_{crit} of S is likely to contain the one and only node, v_i, which has the highest $seed\_ratio_i$.

Efficient and Re-Use Promoting Computation

Given a graph, $G(V, E)$, a seed set, S, and a teleportation probability, β, one way to obtain the RPR-2 scores is to solve the linear optimization problem

$$\texttt{maximize} \quad \sum_{v_i \in S} \vec{\rho_2}[i]$$

subject to the constraints

$$\vec{\rho_2} = (1-\beta)\mathbf{T}_G\ \vec{\rho_2} + \beta\vec{s}, \sum_{v_i \in V} \vec{\rho_2}[i] = 1, \quad 0 \leq \vec{\rho_2}[i] \leq 1,$$

$$\sum_{v_i \in S} \vec{s}[i] = 1, \quad \forall_{v_i \in s} 0 \leq \vec{s}[i] \leq 1, \quad \forall_{v_i \notin s} \vec{s}[i] = 0.$$

While there are many efficient linear solvers that one can use to obtain a solution to the above optimization problem, there are two issues to consider: (a) in general, solving the optimization problem is more expensive than simply solving the linear equations for a given re-start vector $\vec{s}$, and (b) when the seed set S changes (even if the change is small, say one new seed node is considered or one of the seed nodes is dropped) the linear optimization problem needs to be reformulated and solved anew. In this subsection, we note that we can avoid treating the problem as an optimization problem (thereby reducing its cost) and, in the meantime, also support the re-use of existing solutions, by converting the problem into a set of single-seed PPR computations.

Converting the Problem into a Set of Linear Equations.

Given a graph, $G(V, E)$, a seed set, S, and an overall teleportation probability, β, we reformulate the problem (relying on the observation in Section 4.2.3) as follows:

• *Step 1.* for each $v_i \in S$, solve the linear equation $\vec{\pi_i} = (1-\beta)\mathbf{T}_G\ \vec{\pi_i} + \beta\vec{s_i}$, where $\vec{s_i}$ is a re-start vector such that $\vec{s_i}[i] = 1$ and $\forall_{j \neq i}\ \vec{s_i}[j] = 0$;

• *Step 2.* Next, for each $v_i \in S$, compute $\Pi(v_i) = \sum_{v_j \in S} \vec{\pi_i}[j]$;

• *Step 3.* Let S_{crit} be the (small) subset of S, where $S_{crit} = argmax_{v_i} \left(\Pi(v_i)\right)$;

• *Step 4.* If S_{crit} is singleton (i.e., $S_{crit} = \{v_i\}$) then $\vec{\pi_i}$ gives the RPR-2 scores; i.e., $\rho_2 = \vec{\pi_i}$; else (i.e., if S_{crit} is not a singleton), then $\rho_2 = \frac{1}{|S_{crit}|} \sum_{v_i \in S_{crit}} \vec{\pi_i}$.

Note that, since in general the seed set S includes relatively few nodes, the above formulation requires the solution of a small number of single-seed PPR problems. This is especially advantageous when G is large as we can leverage any of the highly effective approximation algorithms [2, 7, 14, 17] or parallelized implementations [8, 53] for computing these PPR scores. Most importantly, the first step of the algorithm (where we solve a linear equation independently for each seed node) can be trivially parallelized by assigning each node to a different computation unit.

Solution Re-Use for Incremental Computation.

Given a graph, $G(V, E)$, a seed set, S, and an overall teleportation probability, β, assume that we have already computed $\vec{\pi_i}$ and $\Pi(v_i)$ for all $v_i \in S$. Let S_{new} be a new seed set, let $\Delta S^+ = S_{new} \setminus S$ denote the new nodes in the seed set and $\Delta S^- = S \setminus S_{new}$ denote the set of nodes dropped from the seed set. We can incrementally compute the RPR-2 scores as follows:

• *Step 1.* For each $v_i \in \Delta S^+$, solve the linear equation $\vec{\pi_i} = (1-\beta)\mathbf{T}_G\ \vec{\pi_i} + \beta\vec{s_i}$, where $\vec{s_i}$ is a re-start vector such that $\vec{s_i}[i] = 1$ and $\forall_{j \neq i}\ \vec{s_i}[j] = 0$;

• *Step 2.* Next, for each $v_i \in \Delta S^+$, compute $\Pi_{new}(v_i) = \sum_{v_j \in S_{new}} \vec{\pi_i}[j]$;

• *Step 3.* Also, for each $v_i \in S_{new} \cap S$, compute $\Pi_{new}(v_i) = \Pi_{new}(v_i) + \sum_{v_j \in \Delta S^+} \vec{\pi_i}[j] - \sum_{v_j \in \Delta S^-} \vec{\pi_i}[j]$;

• *Step 4.* Given these, once again, let S_{crit} be the (small) subset of S, where $S_{crit,new} =$

$argmax_{v_i}\left(\Pi_{new}(v_i)\right)$, and compute the ρ_2 scores as $\rho_2 = \frac{1}{|S_{crit,new}|}\sum_{v_i \in S_{crit,new}} \vec{\pi_i}$.

It is easy to see that, when ΔS^+ and ΔS^- are small, the RPR-2 computations can be done very fast (if necessary, leveraging approximation algorithms [2, 7, 14, 17] and/or parallel implementations [8, 53] for computing the new PPR scores). Once again, the first step of the algorithm (where we solve a linear equation independently for each new seed node) can be trivially parallelized by assigning each node to a different computation unit.

4.2.4 RPR-3: Teleportation-Discounted, Seed-Set Maximal PPR

As we have seen in Section 4.2.2, one disadvantage of the use of standard PPR scores is that the teleportation contribution `t-PPR` score of a seed node (which is exactly $\frac{\beta}{|S|}$ for each seed node) may not capture how significant the node is within the context defined by the entire seed set S. This is especially true in the case of RPR-2 scores, where the set, S_{crit}, of seed nodes selected for re-start is very small. In fact, when S_{crit} is singleton (as most likely), the only seed node in S_{crit} will have at least β overall RPR-2 score. Therefore, when computing the *teleportation-discounted, seed-set maximal PPR scores* (or RPR-3 scores, also denoted as $\vec{\rho_3}$), we replace the use of $\vec{\pi_i}$ vectors for each $v_i \in S$, with the RPR-1 vectors $\vec{\rho_{1,i}}$ as introduced in Section 4.2.2.

4.3 Experimental Evaluation

We ran the experiments on a quad-core Intel(R) Core(TM)i5-2400 CPU @ 3.10GHz machine with 8.00GB RAM. All codes are implemented in Matlab and run using Matlab 7.11.0 (2010b).

4.3.1 Data Sets

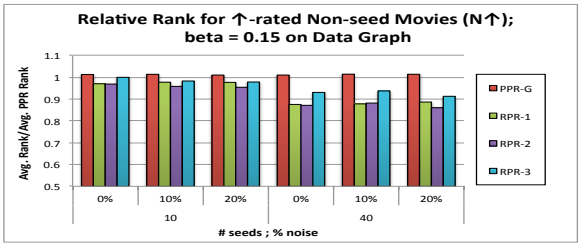
For comparing the different RPR alternatives' performances against conventional PPR, we used the IMDB and MovieLens datasets available from [20, 61]. This dataset contains

Parameter	Range	Default Value
# of seeds	{10, 40}	10
% of noise in the seed set	{0%, 10%, 20%}	10%

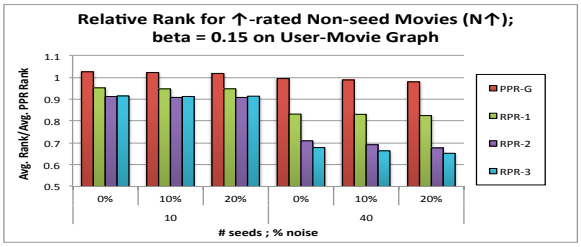
Table 4.3: Personalized PageRank Evaluation Parameters

metadata (e.g. actors, directors) about 1681 movies as well as a total of 100K ratings (between 1, for "dislike" to 5, for "like" provided by 943 users. From this graph, we have constructed three data and social graphs, with distinct semantics and topological properties: (a) *Metadata Graph:* In the metadata graph, nodes represent the data elements (such as movies) and the edges represent relationships between these data elements (such as an actor playing in a movie). The data graph contains 1272 nodes and 60K relationship edges (with average node degree of ∼47). (b) *User-Movie (UM) Graph:* In the UM graph, nodes represent users and movies. There is an edge between a user-movie pair if the user has watched the movie (indicated by the existence of a rating). This graph has 1682 movie nodes, 943 user nodes, and 200K directional (user to movie) edges (with average node degree of ∼76). (c) *Ratings Graph:* The ratings graph consists of the same nodes and edges as the user-movie (UM) graph. However, each ratings edge $u_i \rightarrow m_j$ has an associated numeric weight between 1 and 5, reflecting user u_i's preference for movie m_j. Note that the *metadata graph* captures no knowledge about users and their preferences. The *user-movie (UM) graph*, which can be seen as a rich social network of users and movies, captures which users judged (rated) which movies, but does not capture the value of the rating. The *ratings graph* also captures users' declared preferences in the form of edge weights.

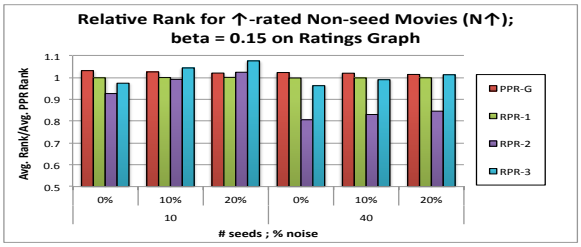
In these experiments, we set the default value of β to 0.15 as is commonly done. Results for other β values are similar.



(a) Metadata graph



(b) User-Movie (UM) graph



(c) Ratings graph

Figure 4.3: Average Ranks of Non-seed Like-rated Movies Relative to Their Ranks Returned by Conventional Ppr (the Ratio for PPR Itself Is 1.0): The Lower Is the Ratio, the Better Is the Measure

4.3.2 Evaluation Strategies

Effectiveness

In order to measure effectiveness of the scores computed for ranking movies, we rely on the following criteria: **Relevance:** If we select a subset of a user's like-rated movies as seeds, the scores should be such that the user's remaining like-rated movies should rank well, whereas user's dislike-rated movies should rank poorly. **Robustness:** Moreover, if the scores are

robust, then even if the seed set contains some small number of dislike-rated movies, this should not negatively affect the rankings significantly.

As shown in Table 4.3, we consider seed sets of different sizes (10 and 40). For each configuration, we select those users who have sufficient ↑-rated movies (e.g., if the target seed set size is 10, then we pick those users with at least 10 ↑-rated movies): For the UM and Ratings graphs, there are 313 users for seed set size 10 and 64 users for seed set size 40. For the Metadata graph, there are 139 users when the seed set size is 10 and 42 users when the seed set size is 40. For each user, we created random seed sets with different degrees of noise (see Table 4.3 for the experiment parameters). Each seed set consists of a number of movies rated "like" by the user (for measuring relevance) and a smaller number of movies rated "dislike" by the same user. The "dislike" movies included in the seed set act as noise (and serve for measuring robustness). For each configuration, we have considered 10 different (randomly picked) seed sets. For each seed set, we treated the rest of the movie ratings by this user as the **ground truth** to help measure the following effectiveness criteria based on the transition probabilities implied by the underlying graph:

• *Recommendation Effectiveness – Average rank for non-seed ↑-rated movies ($AvgRank_{(N\uparrow)}$):* Movies that we know (from the ground truth) that the user would like, but are not included in the seed set are *expected to rank well; i.e., have small average rank values.*

• *Seed Differentiation Power – Average rank for ↓-rated seed movies ($AvgRank_{(S\downarrow)}$):* "Noise" in the seed set (i.e., movies we know from the ground truth that the user does not like, but nevertheless included in the seed set) are *expected to have large average rank values*, even though they are in the seed set. Note that, since a well ranked ↓-rated seed would imply that *the movies neighboring this noisy seed* would also be well ranked, the average rankings of the seed nodes help us (indirectly) quantify the impact of noise on the neighbors of the seeds.

• *Seed-Bias Elimination – Average rank for ↑-rated seed movies ($AvgRank_{(S\uparrow)}$):* Movies that the user likes and are included in the seed set are *expected to rank well and have small average*

rank values.

Efficiency

We compute matrix algebra based formulations of PPR and RPR using Matlab. The RPR-2 and RPR-3 schemes, which seek seed-maximal solutions, are implemented by converting the maximization problem to a set of linear equations (as discussed in Sections 4.2.3 through 4.2.3): these enable the same evaluation mechanism for PPR, PPR-G, and RPR-1 to be applicable also for RPR-2 and RPR-3, making the accuracy and execution time comparisons straightforward.

4.3.3 Results

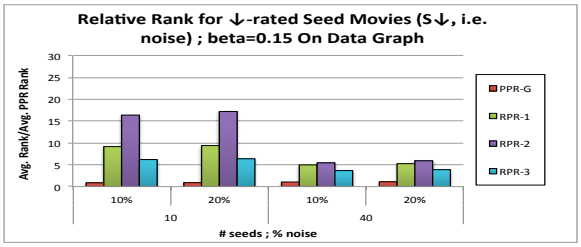
Effectiveness Evaluations

Figures 4.3 through 4.5 compare and evaluate the effectiveness of the different ranking algorithms described in this work, based on the three effectiveness criteria $C = \{N\uparrow, S\uparrow, S\downarrow\}$ listed above. For each of these three criteria, we compare the rankings returned by algorithm A to the rankings returned by the conventional PPR (i.e., PPR with uniform teleportation probabilities) using the measure

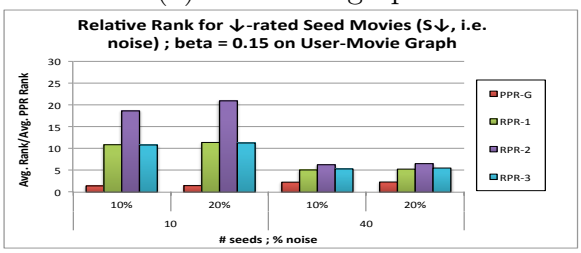
$$relative\_rank(A, C) = (AvgRank_C \text{ by } A)/(AvgRank_C \text{ by } PPR).$$

This measure helps us to observe how algorithm A handles seed noise relative to PPR with no seed differentiation.

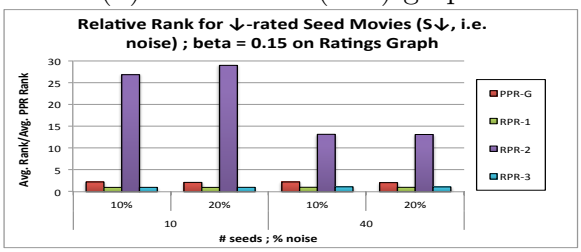
Recommendation Effectiveness: Firstly, let us consider Figure 4.3 which compares the average user rankings of $\uparrow$-rated movies that were not included in the seed set. Since the primary goal of a movie ranking system is to locate non-seed movies that the user would enjoy ($N\uparrow$) and rank them earlier than the other movies, *the smaller the value of the*



(a) Metadata graph



(b) User-Movie (UM) graph



(c) Ratings graph

Figure 4.4: Average Ranks of Dislike-rated Movies Included in the Seed Set Relative to Their Ranks Returned by Conventional PPR (the Ratio for PPR Itself Is 1.0): The Higher Is the Ratio, the Better Is the Measure

relative_rank$(A, N \uparrow)$*, the better would be the recommendations returned by the algorithm* A.

• As we see in the Figure 4.3, PPR-G does not provide any significant advantages over pure PPR, unless the teleportation rate is very small (i.e., not much significance is given to the seed set) or the seed set contains large amounts of noise.

• Figures 4.3 through 4.5 show that the proposed RPR schemes lead to significant improvements in the rankings, with RPR-2 providing the most consistent improvements.

• *Teleportation-discounting* (used in RPR-1 and RPR-3) is effective in (metadata and UM)

graphs, which do not properly capture user preferences. *Seed-set maximization* (used in RPR-2), however, provides benefits for all graphs, including the ratings graph, which reflects the user preferences in the transition probabilities.

• It is important to note that the RPR techniques provide better recommendation rankings even in situations where the seed set contains 0% artificially introduced noise, confirming that RPR provides better personalization given user history.

In addition to the above key observations, we also note that on the UM graph (with higher average node degree), RPR-3 provides the highest improvements. On the metadata graph (with lower average node degree), on the other hand, RPR-3 is less advantageous than RPR-2.

Seed Differentiation Power: Figure 4.4 compares the average user rankings of ↓-rated movies that were included in the seed set. In this case, the *higher the relative rank, the better is the algorithm in differentiating the noise in the seed set.*

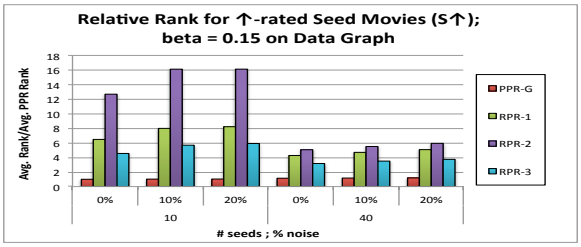
• As we seen in the figure, once again, PPR-G does not provide any significant advantages over pure PPR;

• The proposed RPR algorithms, on the other hand push the ↓ rated seed nodes (i.e., noise) significantly further down in the overall ranking relative to conventional PPR, indicating that RPR algorithms are effective in eliminating seed-bias; and

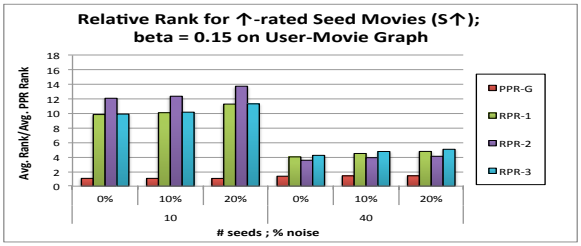
• As before, *teleportation-discounting* (used in RPR-1 and RPR-3) is effective in (metadata and UM) graphs, which do not capture user preferences; but *seed-set maximization* (used in RPR-2), provides benefits for all graphs.

• In fact, comparing the UM and ratings graphs, we see that RPR-2 provides higher average rankings for $S\downarrow$ movies in the ratings graph, indicating that *seed-set maximization* leverages the user preference information embedded in the transition matrix well.

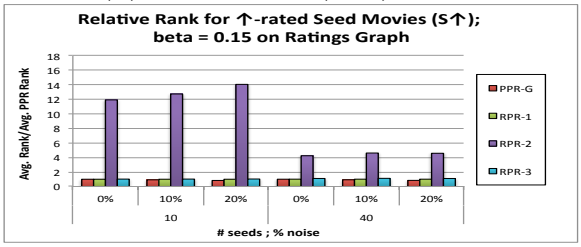
Seed-Bias Elimination: As we discussed in Section 4.2, PPR assumes that all the nodes in the seed set are very important and thus they tend to rank better than most (if not all)



(a) Metadata graph



(b) User-Movie (UM) graph



(c) Ratings graph

Figure 4.5: Average Ranks of Like-rated Movies Included in the Seed Set Relative to Their Ranks Returned by Conventional PPR (the Ratio for PPR Itself Is 1.0): The Lower Is the Ratio, the Better Is the Measure

non-seed nodes. However, *we expect that a seed-bias eliminating ranking system would push some of the highly rated seeds further down in the rankings to bring up those movies that are good, but not used as seeds.*

• In Figure 4.5, we see that this is indeed true for the proposed RPR schemes: as we would expect from a good seed bias eliminating algorithm, expected, the *teleportation-discounting* (for metadata and UM graphs) and *seed-set maximization* (for all graphs) increase the relative rankings of the ↑-rated seed nodes, for accommodating the better rankings of good, but not

seed nodes – as we have already seen in Figure 4.3.

Effectiveness Summary: The above experiments have shown that *teleportation-discounting* is an effective technique in improving ranking effectiveness in graphs which are preference-neutral (like the metadata and user-movie, UM, graphs). The *seed-set maximization* technique, on the other hand, performs well for all graphs (including those that already capture user preferences) and noise scenarios and thus should be the preferred ranking technique (according to the results, even in situations where the noise is 0%).

Efficiency Evaluations

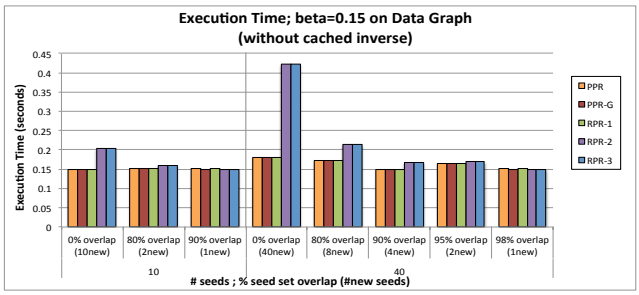
In Figure 4.6 and Figure 4.7, we consider the execution times of the different personalized PageRank algorithms considered in this work (without explicit parallelization). In particular, we consider three scenarios: Fresh graph, fresh seed set: In the first scenario, we are given a graph and a fresh set of seeds and the computation starts from scratch (Figures 4.6(a), (c) and (e), columns corresponding to 0% overlap). Cached graph, fresh seed set: In the second, the graph is fixed and the matrix inverse has already been computed and cached; given a fresh seed set, this cached inverse is used for computing the scores and rankings (Figures 4.7(b), (d) and (f), columns corresponding to 0% overlap). Fresh/cached graph, overlapping seed set: In the third scenario (Figures 4.6 and Figures 4.7 (a) through (f), columns corresponding to more than 0% overlap), the new seed set overlaps with seed sets considered in the past and this overlap is leveraged as described in Section 4.2.3.

• As we see in Figures 4.7(b) and (d), when a cached inverse of the transition matrix is available and the seed set overlap is low, RPR-2 and RPR-3 schemes (which need to solve multiple linear equations per Section 4.2.3) are slower than PPR, PPR-G, and RPR-1 schemes. Thus, when cached inverse is available and seed overlaps are small, we recommend using the RPR-1 scheme which (as we have seen earlier) is more robust than PPR and PPR-G and, also, as

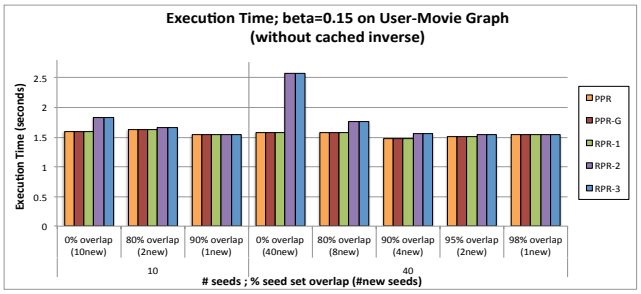
fast.

• The figure also shows that the difference between the various strategies is small or non-existent (a) when the graph itself is fresh (i.e., no cached inverse is available), (b) the seed set is small, or (c) the overlap between the current seed set and the seeds considered in the past is large (i.e., cached solutions for individual seeds can be reused per Section 4.2.3). In these cases, RPR-2 and RPR-3 are competitive in execution times and should be used where the accuracy provided by the *seed-set maximization strategy* is critical.

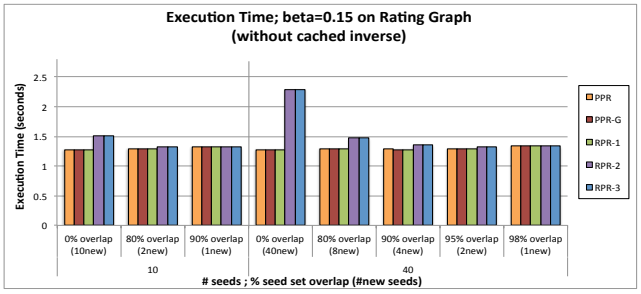
• Also, we would like to remind that, as discussed in Section 4.2.3, RPR is trivially parallelizable by mapping new seeds such that each computation unit processes only one (or few) new seeds (see the cases marked "**1new**" in Figure 4.6 for the impact of processing *only one seed per processing unit*).



(a) Metadata graph, without cached inverse

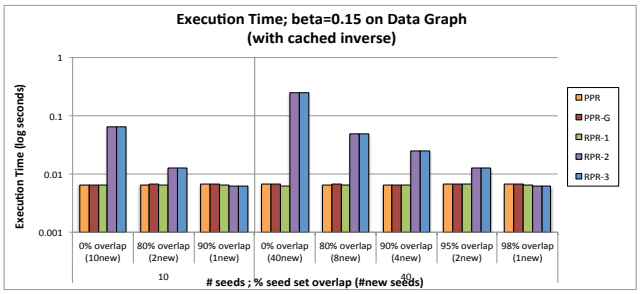


(c) User-Movie (UM) graph, without cached inverse

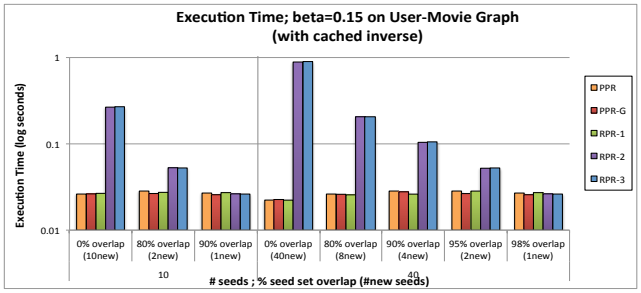


(e) Ratings graph, without cached inverse

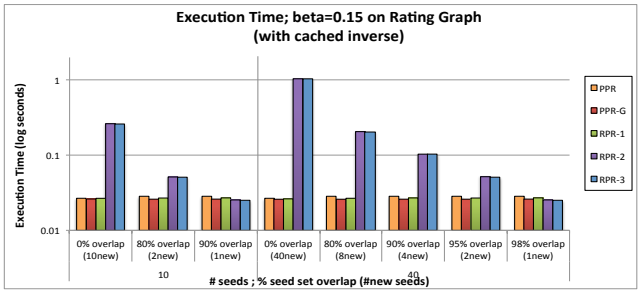
Figure 4.6: Execution Times for Different Measures (w/o Explicit Parallelization): We Consider Situations Where (a,c,e) Personalized PageRank Computation Starts from Scratch and (b,d,f) Where the Cached Matrix Inverses Are Leveraged. For Each Configuration, We Consider Different Rates of Updates to the Seed Set.



(b) Metadata graph, cached inverse



(d) User-Movie (UM) graph, cached inverse (time is log scale)



(f) Ratings graph, cached inverse

Figure 4.7: Execution Times for Different Measures (w/o Explicit Parallelization): We Consider Situations (b,d,f) Where the Cached Matrix Inverses Are Leveraged. For Each Configuration, We Consider Different Rates of Updates to the Seed Set.

Chapter 5

BICP: BLOCK-INCREMENTAL CP DECOMPOSITION WITH UPDATE SENSITIVE REFINEMENT

5.1 Introduction

With many applications relying on multi-dimensional datasets for decision making, tensors (or multi-dimensional arrays) are emerging as a popular data representation [52, 31, 30]. 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 2.5) and Tucker decompositions [54]. These form the basis for many data analysis and knowledge discovery tasks, from clustering, trend and anomaly detection [31] to correlation analysis [47].

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 work, 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.

5.1.1 Key Observations

[34] and [39] presented a block-based alternative to tensor decomposition: (a) in their first phase, they 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 work, 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. As shown in Figure 2.3:

- In Phase 1, each block is decomposed; consequently, there is no need to recompute unaffected blocks' decompositions.
- In fact, if the update on an effected block is relatively small, we may be able to completely avoid re-decomposing this block and, instead, we can maintain the corresponding sub-tensor incrementally.
- 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 work, a subset of those blocks may often be sufficient).

5.1.2 Contributions: Two-Phase Block Incremental Tensor 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. $\mathring{\boldsymbol{\mathcal{X}}}$. Based on the above observations, in this work, 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.

5.2 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:

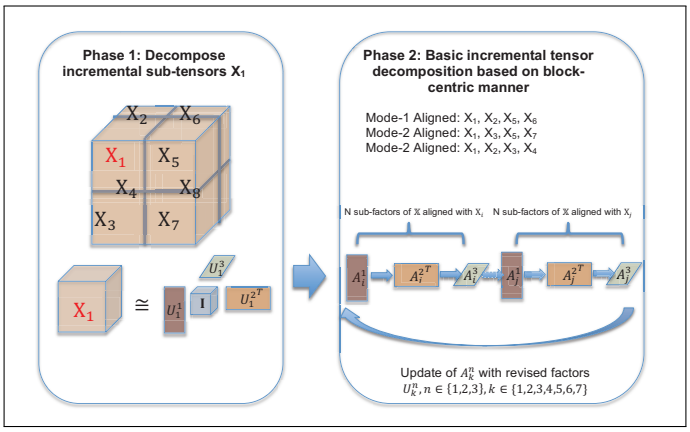


Figure 5.1: Illustration of Basic Method of Incremental Block-based Tensor Decomposition(the Notation Is Introduced in Section 5.2)

• **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 $\boldsymbol{U}_{\vec{k}}^{(1)}, \boldsymbol{U}_{\vec{k}}^{(2)}, \ldots, \boldsymbol{U}_{\vec{k}}^{(N)}$. 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 5.1: 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 $\boldsymbol{U}_{(1)}^{(1)}$, $\boldsymbol{U}_{(1)}^{(2)}$, $\boldsymbol{U}_{(1)}^{(3)}$ 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 5.2.1, 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.

• **Observation #2 (Update Sensitive Refinement in Phase 2):** A close look at Phase 2 of Algorithm 2 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 $\boldsymbol{\mathcal{X}}_{\vec{k}}$:

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

This is visualized in Figures 2.3 and 5.1. Let $\boldsymbol{\mathcal{X}}_1$ be the modified sub-tensor in Figure 5.1: 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 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.

5.2.1 Optimization #1 - Incremental Factor Maintenance in Phase 1

The basic BICP algorithm presented in Algorithm 4 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 when the updates are small.

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 [9] 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 [41] or LWI-SVD [15]. 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 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 [45] algorithm applies this idea to maintain factors of an evolving tensor. The authors also suggest that one can sample fibers with high norms, with relatively high impact on the decomposition, to reduce execution time. In this work, 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 5.3, 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 5.3, focusing only on matricization columns that carry most of the energy of the update matrix, Δ_t, (see Steps 1 and 2c of Algorithm 5), 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). We can show the computational complexity of Algorithm 3 is $\mathcal{O}(\sum_{i=1}^{N} |\mathcal{C}| \times F \times \frac{I_i}{K_i})$ where N is the number of modes, F is the target decomposition rank, $|\mathcal{C}|$ is the number of update-critical fibers, I_i is the dimensionality of the tensor of mode i, and K_i is the number partitions of that mode.

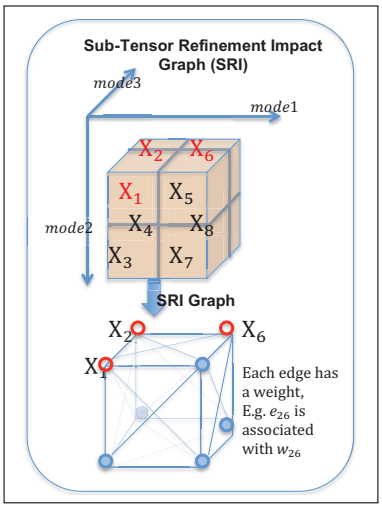


Figure 5.2: Illustration of the Sub-tensor Refinement Impact (SRI) Graph Construction

5.2.2 Optimization #2- Reducing Redundant Refinements in Phase 2

The basic BICP method presented in Algorithm 4 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.

As we have discussed earlier in Section 5.2, 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. The observations are aligned with the problems discussed in Chapter 3 that sub-tensor impact graph (SIG) is proposed to quantify how much an update on a sub-tensor impacts sub-factors on other sub-tensors, then we could use SIG and corresponding strategies to optimize Phase 2.

Computing Sub-Tensor Impact Scores Relative to the Tensor Updates

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.

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.

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 2.2. The corresponding SRI graph is a directed, weighted graph, $G(V, E, w())$, the graph construction details are described in Section 3.2. We use value alignment defined in Section 3.2.2 to assign edge weight and account for error propagation among sub-tensors: 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 5.2.1 *We visualize this in Figure 5.2. 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 5.2.2.*
- *Then, we calculate the impact score using Personalized PageRank(PPR) Equation introduced in Section 5.2.2. 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 5.2.2 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 refinements propagate within G including both *direct* and *indirect* refinements. In particular, we rely on personalized PageRank (PPR [6])

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. 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 $I_\Delta(\boldsymbol{\mathcal{X}}_{\vec{k}}) = p\vec{p}r[\vec{k}]$. 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.

Phase-2 Optimization Based on SRI

According to Section 3.3, I proposed three optimization strategies that rely on impact score calculated from SIG. Given same property can be applied here, we applied the three optimizations as:

- *Optimization 2-I:* Ignore low impact sub-tensor in the refinement based on impact score from Section 3.3.4.
- *Optimization 2-P:* Assign probability of refinement based on impact score from Section 3.3.5.
- *Optimization 2-R:* Assign decomposition rank based on impact score from Section 3.3.3.

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

5.2.3 Optimized BICP Algorithm, $BICP_{opt}$

Algorithm 6 presents the optimized BICP algorithm, which combines the optimizations presented above. In the next section, we experimentally investigate the performances of these optimizations. Phase-1 complexity is reported in Section 5.2.1. By extending the complexity formulation from [39], we can obtain the complexity[1] of Phase 2 as

$$\mathcal{O}((F \times \sum_{i=1}^{N} \frac{I_i}{K_i} + F^2) \times \mathcal{R} \times \mathcal{H} \times |\mathcal{D}|)$$

where $\mathcal{R}$ 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 partitions along that mode.

5.3 Experiments

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

5.3.1 Experiment Setup

Data Sets. In these experiments, we used three datasets: *Epinions* [60], *Ciao* [60], and *Enron* [59]. 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

[1]Here we report the complexity of $2-I$ and other refinement method complexity can be derived similarly.

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.

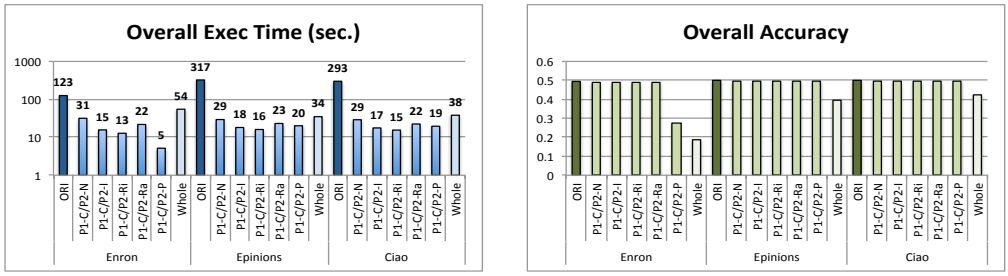
Alternative Strategies. In this section, we consider the following strategies to maintain the tensor decomposition:

In the first strategy, (a) we apply the basic two-phase block-centric decomposition strategy in Algorithm 2; 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`).

In addition, we consider several configurations of BCIP. For Phase 1, there are four options: (b) to redo the tensor decomposition for the updated sub-tensors (`P1N`), (c) to utilize incremental tracking algorithm: STA with updating fibers on all modes (`P1A`), (d) STA with updating fibers that are update-critical with highest energy among the all the affected fibers (`P1C`, default) or (e) STA with updating fibers with max norms (`P1M`). For Phase 2, again, we have several alternatives: (f) applying Phase 2 without any impact score based optimization (`P2N`), (g) ignoring $L\%$ of sub-tensors with the lowest impact scores (`P2I`), and (h) reducing the decomposition rank of sub-tensors (`P2Ra` and `P2Ri`), or (i) 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 $p = 0.1$.

In addition the block-based BCIP and its optimizations, we also considered, as an efficient alternative, (j) application of the incremental factor tracking process (Algorithm 3) to the whole tensor as in STA [45] – in the charts, we refer to this alternative approach as `Whole`.

Evaluation Criteria. We report decomposition accuracy and decomposition time (Phase 1, Phase 2, and total) for different settings. In these experiments, the target decomposition



(a) Execution times (b) Decomposition accuracies

Figure 5.3: Comparison of (a) Execution Times and (B) 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

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].

5.3.2 Discussions of the Results

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

General Overview

Figure 5.3 compares execution times and accuracies of several approaches. Here, `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 (`P1C/P2N`, `P1C/P2I`, `P1C/P2Ri`, `P1C/P2Ra`, `P1C/P2P` all correspond to different optimizations of the proposed BICP approach, with incremental factor tracking, `P1C`, in Phase 1 and five different

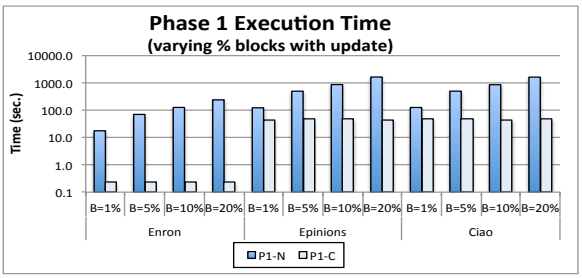


Figure 5.4: Phase 1 Execution Time, With and Without Incremental Factor Tracking (I.E., `P1N` Vs `P1C`), As a Function of the Percent of Blocks With Updates

strategies (`P2N`, `P2I`, `P2Ri`, `P2Ra`, `P2P`) for Phase 2.

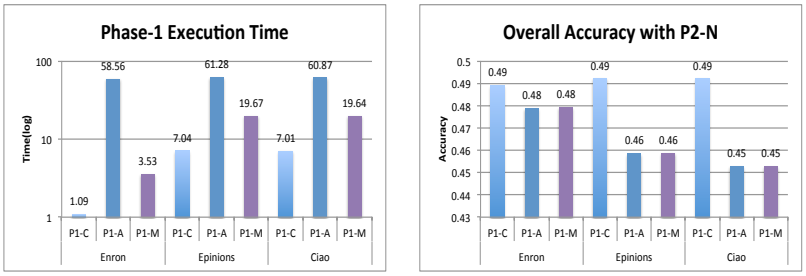
Firstly, a quick look at this figure shows that the two social media data sets, Epinions and Ciao, with similar sizes and densities show very similar execution time and accuracy patterns. The figure also shows that the Enron data set also exhibits a pattern roughly similar to the other data sets, despite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

The key observation in Figure 5.3 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.

Evaluation of Phase-1 Optimizations

Figure 5.4 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, the overall



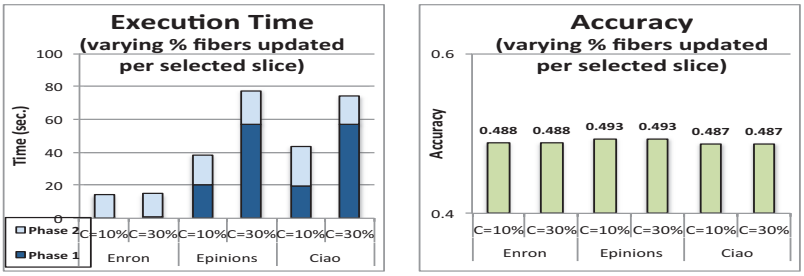
(a) Exec. time (b) Accuracy

Figure 5.5: (A) Execution Time and (B) Accuracy Compare the Performance of Different Incremental Factor Maintenance Method in Phase 1

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.

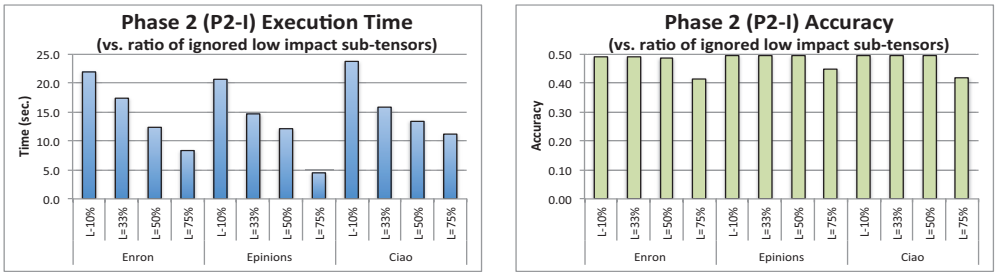
As we expected, in Figure 5.5, `P1C`, which focuses on fibers that are update-critical (with highest energy among the all the affected fibers), achieves the best execution time as well as accuracy while `P1A` performs the worst in time. It considers all the modes matricizations during tracking process. It also results in lower accuracy due to the imperfections in the factor tracking process: revising factor matrices for all fibers without considering their energies can potentially introduce errors in the process. `P1M` has better time result than `P1A` since it only updates fibers with max norm while achieving similar accuracy result with `P1A`.

Figure 5.6 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.



(a) Exec. time (b) Accuracy

Figure 5.6: (A) Execution Time and (B) Accuracy as the Ratio of the Modified Fibers Vary



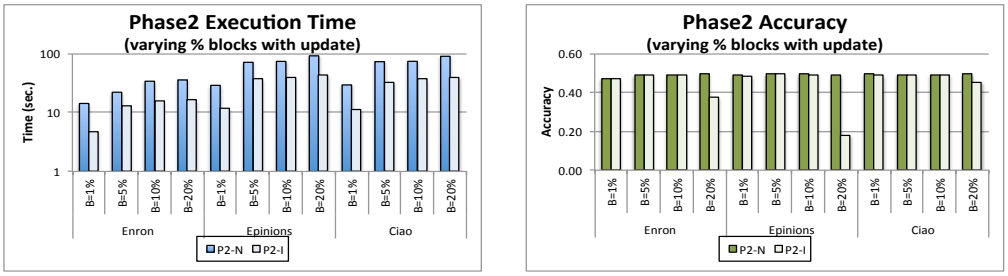
(a) Execution times (b) Decomposition accuracies

Figure 5.7: Phase 2 (A) Execution Time and (B) Accuracy for `P2I`, When Varying the Ratio of the Low Impact Sub-tensors Ignored During Refinement

Evaluation of Phase-2 Optimizations

The `P2I` optimization for Phase 2 involves ignoring low impact sub-tensors during the refinement process. In Figure 5.3, we had seen that `P2I` provides significant gains over the non-optimized Phase 2, `P2N`. As we see in Figure 5.7, ignoring such low impact sub-tensors can indeed save significant work. Moreover, unless the ratio of the ignored tensors is very high (75%, in the considered default scenario), these gains are obtained without accuracy penalties.

Figure 5.8 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



(a) Execution times (b) Decomposition accuracies

Figure 5.8: 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 5.1: Impact of Different Low-Impact Sub-Tensor Refinement Probabilities in `P2P` (for Enron)

updates, with and without the ignoring of low impact 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 imposing fixed ratio of ignored blocks.

To see the impacts of the `P2P`, `P2Ri` and `P2Ra` optimization, we again consider Figure 5.3: in this figure, we see that `P2Ri`, 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, `P2Ri` provides the best execution time, with no accuracy penalty. As we expected `P2Ra` is slower than `P2Ri` because it consider all relevant sub-tensors.

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 5.1: 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 P2Ri optimization, which provides execution time gains, without any reduction in accuracy.

Algorithm 4 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) $\mathcal{D} = \bigcup_{\vec{t} \in \mathcal{T}} direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{t}})$
   (b) Repeat for $\vec{d} \in \mathcal{D}$
      i. for each mode $1 \leq m \leq N$ for which $\vec{d}$ is aligned with any $\vec{t} \in \mathcal{T}$
         A. update $\boldsymbol{A}^{(m)}_{(d_m)}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$, for each block $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,d_m,*,\ldots,*]} \in \mathcal{D}$; more specifically,
            - compute $\boldsymbol{T}^{(m)}_{(d_m)}$, which involves the use of $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$ (i.e. the mode-i factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,d_m,*,\ldots,*]}$)
            - revise $\boldsymbol{P}_{[*,\ldots,*,d_m,*,\ldots,*]}$ and $\boldsymbol{Q}_{[*,\ldots,*,d_m,*,\ldots,*]}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$ and $\boldsymbol{A}^{(m)}_{(d_m)}$
            - compute $\boldsymbol{S}^{(m)}_{(d_m)}$ using the above
            - update $\boldsymbol{A}^{(m)}_{(d_m)}$ using the above
         B. for all $\vec{l} = [*, \ldots, *, d_i, *, \ldots, *] \in \mathcal{D}$
            - update $\boldsymbol{P}_{\vec{l}}$ and $\boldsymbol{Q}_{\vec{l}}$ using
              – $\boldsymbol{U}^{(m)}_{\vec{l}}$ and $\boldsymbol{A}^{(m)}_{(d_m)}$

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

Algorithm 5 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}

Algorithm 6 Optimized block-centric incremental tensor decomposition

Input: original tensor, $\boldsymbol{\mathcal{X}}$; 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;

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) $\mathcal{D} = \bigcup_{\vec{t} \in \mathcal{T}} direct\_impact(\boldsymbol{\mathcal{X}}_{\vec{t}})$
   (b) Get refinement impact score, $I_{\Delta}(\vec{d})$, for all $\vec{d} \in \mathcal{D}$
   (c) If 2-Ra
      - $F_{\vec{d}} = F \times \frac{PPR(\boldsymbol{\mathcal{X}}_{\vec{d}})}{max\{PPR(\mathcal{D})\}}$
      - Truncate $U_{\vec{d}}$ and $P_{\vec{d}}$ according to $F_{\vec{d}}$
   (d) If 2-N, 2-I, 2-Ri or 2-P
      i. For the lowest $L\%$ of sub-tensors $\vec{d} \in \mathcal{D}$, $F_{\vec{d}} = F$
         A. if 2-I or 2-Ri, then $p_{\vec{d}} = 0.0$
         B. if 2-P, then $p_{\vec{d}} = p_{low}$
      ii. For the highest $(100 - L)\%$ of sub-tensors in $\vec{d} \in \mathcal{D}$,
         A. $p_{\vec{d}} = 1.0$
         B. if 2-I or 2-P, then $F_{\vec{d}} = F$
         C. if 2-Ri, then
            - $F_{\vec{d}} = F \times \frac{PPR(\boldsymbol{\mathcal{X}}_{\vec{d}})}{max\{PPR(\mathcal{D})\}}$
            - Truncate $U_{\vec{d}}$ and $P_{\vec{d}}$ according to $F_{\vec{d}}$
   (e) Repeat for $\vec{d} \in \mathcal{D}$
      - With probability $p_{\vec{d}}$ do
         i. for each mode $1 \leq m \leq N$ for which $\vec{d}$ is aligned with any $\vec{t} \in \mathcal{T}$
            A. update $\boldsymbol{A}^{(m)}_{(d_m)}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$, for each block $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,d_m,*,\ldots,*]} \in \mathcal{D}$; more specifically,
               – compute $\boldsymbol{T}^{(m)}_{(d_m)}$, which involves the use of $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$ (i.e. the mode-i factors of $\boldsymbol{\mathcal{X}}_{[*,\ldots,*,d_m,*,\ldots,*]}$)
               – revise $\boldsymbol{P}_{[*,\ldots,*,d_m,*,\ldots,*]}$ and $\boldsymbol{Q}_{[*,\ldots,*,d_m,*,\ldots,*]}$ using $\boldsymbol{U}^{(m)}_{[*,\ldots,*,d_m,*,\ldots,*]}$ and $\boldsymbol{A}^{(m)}_{(d_m)}$
               – compute $\boldsymbol{S}^{(m)}_{(d_m)}$
               – update $\boldsymbol{A}^{(m)}_{(d_m)}$
            B. for all $\vec{l} = [*, \ldots, *, d_i, *, \ldots, *] \in \mathcal{D}$
               – update $\boldsymbol{P}_{\vec{l}}$ and $\boldsymbol{Q}_{\vec{l}}$ using $\boldsymbol{U}^{(m)}_{\vec{l}}$ and $\boldsymbol{A}^{(m)}_{(d_m)}$

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

Chapter 6

FOCUSING DECOMPOSITION ACCURACY BY PERSONALIZING TENSOR DECOMPOSITION (PTD)

6.1 Introduction

Tensors are multi-dimensional arrays and are commonly used for representing multi-dimensional data, such as Web graphs, sensor streams, and social networks. Specific uage examples in the literature include representations of RDF triples of the form (subject-predicate-object) in knowledge bases (venue-author-keywords) relationships in scientific digital libraries [31], and (movie-user-rating) relationships in movie recommendation [20]. As a consequence, tensor decomposition operations (such as CP [19] and Tucker [54]) began to form the basis for many data analysis and knowledge discovery tasks, from clustering, trend detection, anomaly detection [31], to correlation analysis [47].

One key problem with tensor decomposition, however, is its computational complexity – especially for dense data sets, the decomposition process takes exponential time in the number of tensor modes. While, the process is relatively faster for sparse tensors, decomposition is still a major bottleneck in many applications: decomposition algorithms have high computational costs and incur large memory overheads and, thus, are not suitable for large problems. Recent improvements (including scalable implementations, such as TensorDB [28, 29], Grid PARAFAC [39], and GigaTensor [25]) also suffer from high computational costs.

6.1.1 Accuracy vs. Decomposition Time

Due to the approximate nature of the tensors decomposition process, one way to reduce computational requirements might be to trade performance with accuracy. However, natu-

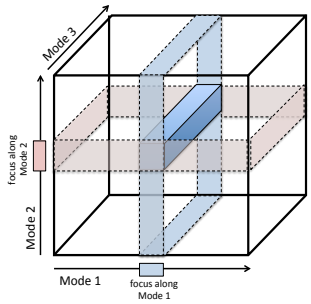


Figure 6.1: Personalization Through User's Focus of Interest: In This Example, the User Has Expressed Special Interest on a Subrange of Values along the First and Second Modes of the Tensor. Consequently, Any Processing on the Tensor (Including Tensor Decompositions) Should Preserve the Accuracy along the Corresponding Slices and Especially the Region Consisting of the Intersection of Foci of Interest

rally, a drop in accuracy may not be acceptable in many applications. Therefore, this is not a feasible solution to tackle the computational cost.

In this work, we first recognize that in many 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, the user may not be interested in maintaining high accuracy. For example, in a clustering application, the user might be interested in ensuring the clustering accuracy (i.e., differentiating power) for a high priority subset of the objects in the database. As a specific example, consider a movie recommendation application, where tensor decomposition of a *movie-user-ratings* tensor is used for generating recommendations: in this example, the user may want to decompose this tensor in a way that maximizes accuracies for *recent movies* and the *most active users* of the system.

6.1.2 Contribution of this Paper: Personalized Tensor Decomposition (PTD)

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.

6.1.3 Problem Formulation

Let an N-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, be partitioned into a set (or grid) of sub-tensors $\boldsymbol{\mathfrak{X}} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ where, K_i indicates the number of partitions along mode-i, the size of the j^{th} 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, $\|\boldsymbol{\mathfrak{X}}\|$, 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 $\boldsymbol{\mathcal{X}}$ is re-ordered before it is partitioned n 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[1].

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

Given the above, let us use $\boldsymbol{\mathcal{X}}_P$ as a shorthand to denote the cells of $\boldsymbol{\mathcal{X}}$ 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* $\hat{\boldsymbol{\mathcal{X}}}$ of $\boldsymbol{\mathcal{X}}$ in such a way that

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

where $\hat{\boldsymbol{\mathcal{X}}}_P$ is the reconstruction of the user selected region from the personalized decomposition $\hat{\boldsymbol{\mathcal{X}}}$ and $\tilde{\boldsymbol{\mathcal{X}}}_P$ is the reconstruction of the same region from a decomposition of $\boldsymbol{\mathcal{X}}$ 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 $\hat{\boldsymbol{\mathcal{X}}}$ will be lesser than the time needed to obtain preference insensitive decomposition $\tilde{\boldsymbol{\mathcal{X}}}$ and also that the personalized decomposition minimally impacts the rest of the tensor; i.e.,

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

6.1.4 Sub-Tensor Impact Graph Application and Optimization

Remember from Section 2.2, 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. PTD leverage the block-based tensor decomposition and construct sub-tensor impact graph to account for the error propagation. Algorithm 7 present an overview of PTD process. Specifically, there are three major steps

- Step-1: compute sub-tensor impact graph
- Step-2: assign an initial decomposition rank based on sub-tensor impact graph and user focus sub-tensors
- Step-3: obtain a block-based CP decomposition based on sub-tensor decomposition

In the following section, we focus on expanding step-2 and step-3 of PTD process for the optimization strategies.

Sub-Tensor Impact Graph Weight Assignment

After constructing SIG, one of the major concern is how to assign edge weight to better account for inaccuracy propagation among sub-tensors. PTD adopts four proposed strategies in Section 3.2.2:

- *Uniform Edge Weights:* Uniform edge weight assigns edge weight 1 to each edge.
- *Surface of Interaction based Weights:* the uniform edge weight may not account for the impact of the varying dimensions of the sub-tensors on the error propagation. We propose surface of interaction based weights to account for the varying shapes and sizes of the sub-tensors.
- *Value Alignment based Edge Weights:* Although surface of interaction based edge weights they fail to take into account for data distribution with in the subtensor. they ignore how the values within the sub-tensors are distributed and whether these distributions are aligned across them.
- *Combined Edge Weights:* A potentially more effective alternative would be to combine the surface of interaction and value alighment based weightes into a single weight which take account both aspects

Rank Assignment for Personalized Tensor Decomposition

Remembered from Section 3.3.1, one critical observation is *sub-tensor rank flexibility* that sub-tensors are not required to have the same decomposition target rank. However, the key difficulty is that one cannot arbitrarily reduce the decomposition ranks of low priority partitions because the accuracy in one partition may impact final decomposition accuracy of other tensor partitions. As visualized in Algorithm 7, the proposed PTD algorithm first constructs a *sub-tensor impact graph*, G, that accounts for the propagation of inaccuracies

Algorithm 7 Overview of the PTD-CP process

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

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

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

2- for each partition $\vec{k} \in \mathcal{K}$, assign an initial decomposition rank $F_{\vec{k}}$ 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, $\hat{\boldsymbol{\mathcal{X}}}$, of $\boldsymbol{\mathcal{X}}$ using the partitioning pattern $\mathcal{K}$ and the initial decomposition ranks $\{F_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$;

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

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 $\boldsymbol{\mathcal{X}}_P$; i.e., the cells of $\boldsymbol{\mathcal{X}}$ collectively covered by the user's declaration of interest (i.e., $\mathcal{K}_P$).

Therefore. relied on this observation, PTD adopt optimization-1 proposed in Section 3.3.3 to assign decomposition rank based on impact score. Intuitively, when we apply optimization-1, user focus sub-tensors are used as seed node to compute Personalized PageRank score (impact score) based on SIG. Then sub-tensor target ranks are assigned based on impact score such that highest accuracy impact on the set of sub-tensors are chosen by the user whereas other sub-tensors are assigned progressively smaller ranks

6.2 Experimental Evaluation

In this section, we report experiments 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 user. We also evaluate different strategies for accounting the propagation of inaccuracies within the tensor during the PTD process.

6.2.1 Experiment Setup

Data Sets. In these experiments, we used three real datasets: *Epinions* [60], *Ciao* [60], and *Enron* [59]. The first two of these are comparable in terms of their sizes and semantics: they are represented in the form of $170 \times 1000 \times 18$ (density 2.4×10^{-4}) and $167 \times 967 \times 18$ (density 2.2×10^{-4}) tensors, respectively, and both have the schema $\langle user, item, category \rangle$. The *Enron* email data set, however, is much larger ($5632 \times 184 \times 184$, density 1.8×10^{-4}) and has a different schema, $\langle time, from, to \rangle$.

User Focus. In order to observe the impacts of different types of user feedback, we considered (a) different numbers of partitions, (b) different ratios of partition sizes, and (c) different numbers of partitions included in the user's focus. Table 6.1 lists the various partitioning scenarios considered in these experiments. For each partitioning scenario, random partitions of the input tensors have been created and we report averages of the runs for different focus selections on these partitions. In the experiments, we also considered scenarios where the user's interest (d) is focused on different numbers $(1, 2, 3, 4)$ of sub-tensors.

Decomposition Strategies. We considered five block-based tensor decomposition strategies: not personalized block-based (`NP`[2]), uniform edge weights (`UNI`), surface of interaction

[2]Results for CP ALS based PARAFAC were similar to block based `NP` and thus are omitted due to space limits.

$2 \times 2 \times 2$ 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%

$4 \times 4 \times 4$ Partitions				
Config.	*Part.*#1	*Part.*#2	*Part.*#3	*Part.*#4
1- Most Bal.	25%	25%	25%	25%
2-	10%	20%	30%	40%
3-	1%	10%	40%	49%
4- Least Bal.	1%	5%	25%	69%

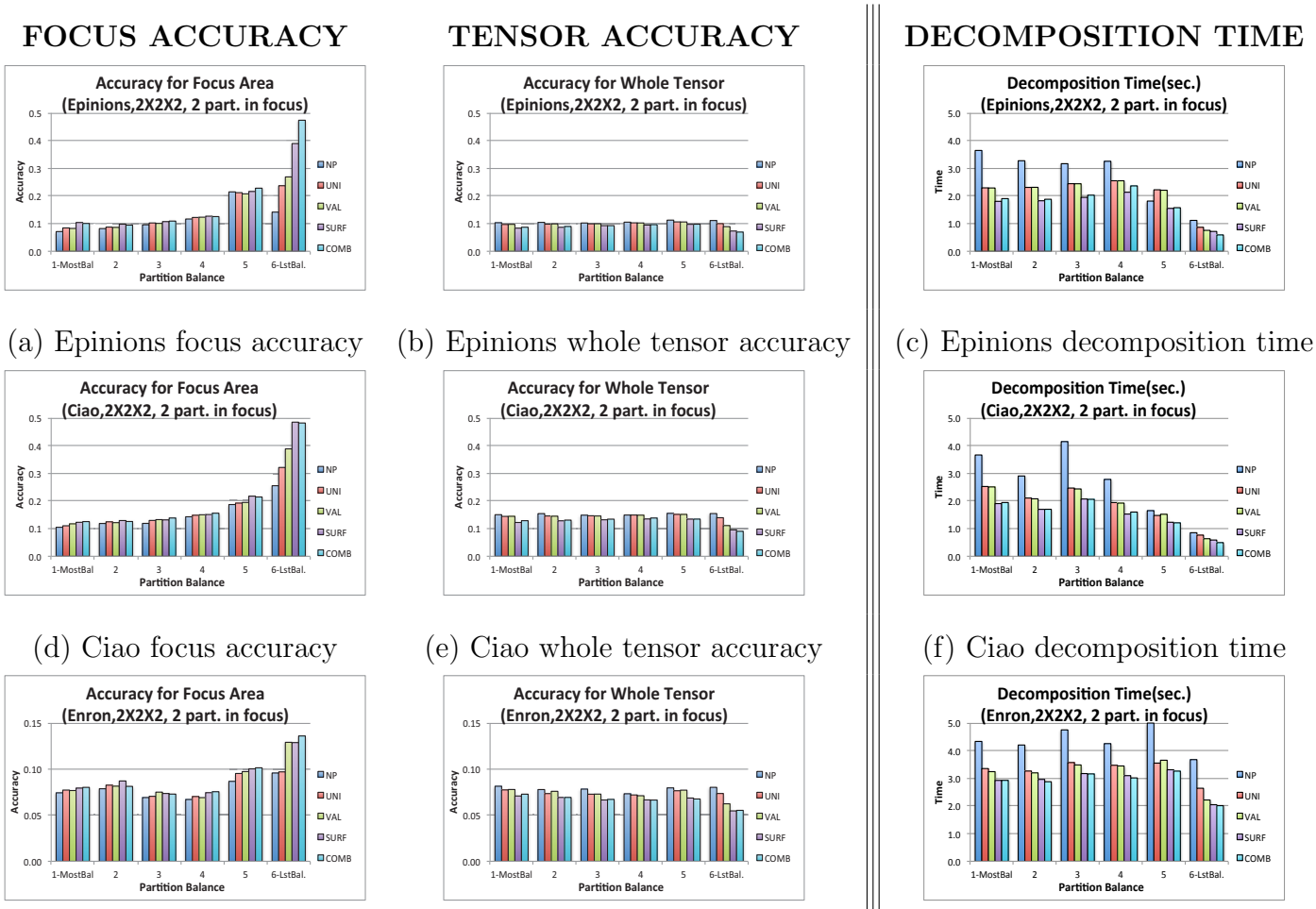
Table 6.1: Various Tensor Partitioning Scenarios Considered: Percentages Are the Sizes of the Partitions (Relative to the Size of the Mode) along Each Mode

based edge weights (`SURF`), value alignment based edge weights (`VAL`), and the combined edge weights (`COMB`). These were implemented by modifying the grid PARAFAC technique [39]. The decomposition rank, F, is set to 10.

Evaluation Criteria. We report decomposition accuracy, in particular, accuracies for both user's area of focus as well as the whole tensor. We also report the decomposition time for different decomposition schemes[3].

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].

[3]Note that the time to compute rank assignments for the partitions was negligible (e.g. on the order of few milliseconds for $4 \times 4 \times 4$ partitioning of the tensor), therefore we do not report and investigate rank assignment times explicitly.



(a) Epinions focus accuracy (b) Epinions whole tensor accuracy (c) Epinions decomposition time

(d) Ciao focus accuracy (e) Ciao whole tensor accuracy (f) Ciao decomposition time

(f) Enron focus accuracy (g) Enron whole tensor accuracy (h) Enron decomposition time

Figure 6.2: Experiment Results for $\mathbf{2 \times 2 \times 2}$ Partitioning with 2 Partitions in Focus

6.2.2 Discussion of the Results

General Overview. Figure 6.2 shows accuracies (for the focus area as well as for the whole tensor) and decomposition times for the configuration with $2 \times 2 \times 2$ partitioning of the input tensor and 2 partitions highlighted in the user focus.

As we see in this figure, as expected, for all data sets, 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).

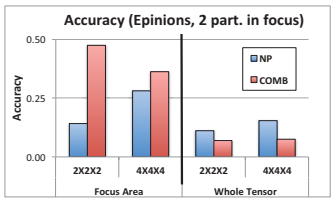
While, as would be expected, the PTD algorithms impact the overall decomposition accuracy for the whole tensor, this is more than compensated by gains in accuracies in high-priority areas [4]. Moreover, the figure also shows that the gains in accuracy in high-priority partitions within the user's focus comes also with significant gains in execution times for the decomposition process.

As a control, we have also considered a naive approach where we simply zero-out out-of-focus areas of the input tensor: As would be expected, this had drastic impacts on the out-of-focus areas. For example, for the Epinions data sets, the out-of-of-focus accuracies for this zeroing-out method were 1.82E-05 (for the most balanced partitioning), 2.20E-05, 2.44E-05, 2.40E-05, 1.54E-05, 3.99E-08 (for the least balanced partitioning); i.e., several orders worse than `COMB`.

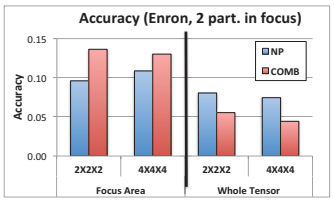
The figure also establishes that, both in terms of accuracy and execution time gains, we can order the various edge weighting strategies as follows: `UNI` (least effective), `VAL`, `SURF`, and `COMB` (most effective). In other words, as we argued in Section 3.2.2, the most effective way to account for 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.

Varying the Number of Partitions. Figure 6.3 verifies whether the observations also hold when the number of partitions of the tensor is larger (i.e., $4 \times 4 \times 4$ vs. $2 \times 2 \times 2$). Due to the large number of alternatives, for the $4 \times 4 \times 4$ partitioning scenario, the two partitions in the user focus are generated as follows: for each of the 64 partitions, 10 other partitions are paired with it randomly, leading to 640 pairs of partitions used as user focus (consequently,

[4] Accuracies for the out-of-focus areas are lower-than but similar to the whole tensor accuracies reported in the figure: For example, for the Epinions data set the whole tensor accuracies for `COMB` are (from the most to least balanced configurations) 0.095, 0.095, 0.098, 0.100, 0.104, 0.069. For the same scenario, out-of-focus accuracies are 0.079, 0.089, 0.097, 0.097, 0.095, and 0.051.



(a) Epinions accuracy



(b) Enron accuracy

Decomposition time gains $(\frac{\text{time}_{\text{NP}} - \text{time}_{\text{COMB}}}{\text{time}_{\text{NP}}})$		
Data Set	$2 \times 2 \times 2$	$4 \times 4 \times 4$
Epinions	47%	80%
Enron	45%	75%

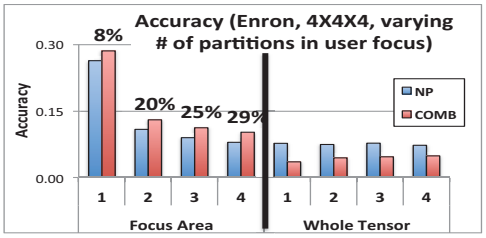
Figure 6.3: Experiment Results for $\mathbf{2 \times 2 \times 2}$ vs. $\mathbf{4 \times 4 \times 4}$ Partitioning with 2 Partitions in Focus

each partition is guaranteed to be in user focus at least 10 times).

As we see in Figure 6.3, PTD based algorithms boost the accuracy for the high priority regions of the tensor. Moreover, the *decomposition time gain* table shows that the time gains, defined as $\frac{time_{NP} - time_{COMB}}{time_{NP}}$, further improves as the number of partitions of the input tensor increases.

Varying the Number of High-Priority Partitions. Figure 6.4 studies the impact of the number of partitions that are in the user focus. Due to the large number of alternatives, the k partitions (for $1 \leq k \leq 4$) in the user focus are generated as follows: each of the 64 partitions for the $4 \times 4 \times 4$ partitioning scenario has been paired with 10 $(k-1)$ partitions selected randomly, leading to a total of 640 partition k-tuples that are used as high-priority partitions in the user focus (consequently, each partition of the tensor is guaranteed to be in user focus at least 10 times).

As we see, the PTD based algorithms boost the accuracy in the high priority region for all user foci considered: the relative gains in accuracy improves from 8% for a single high-priority partition to 29% for 4 high-priority partitions in focus. Naturally, the gains vary depending



(a) Enron accuracy

Decomposition time gains $(\frac{time_{NP}-time_{COMB}}{time_{NP}})$				
Data Set	*Number of Partitions in Focus*			
	1	2	3	4
Epinions	86%	80%	76%	75%
Enron	81%	75%	70%	69%

Figure 6.4: Results for Varying Number of Partitions in Focus (Least Balanced Partitioning Configuration for $4 \times 4 \times 4$; Also Ciao Results and Epinions Accuracy Results Are Omitted Due to Space Constraints)

on the distributions of the values in the tensors at different scales. The *decomposition time gain* remains high as the number of high-priority partitions increase.

Chapter 7

DPTD: DYNAMIC EVOLVING USER FOCUS BLOCK TENSOR DECOMPOSITION

7.1 Introduction

Nowadays, many applications leverage multi-dimensional datasets for decision making. For example in computer vision and graphics, we have large amount of high-dimensional data such as images, videos and also medical data. As we discussed in earlier works, the multi-dimensional array is known as tensor and tensor decomposition is commonly used for data analysis and knowledge discovery tasks, from clustering, trend and anomaly detection [31] to correlation analysis [47]. Extended from block-based tensor decomposition, in [34], we proposed a two-phase block-centric CP decomposition (2PCP) shown in Algorithm 2. In phase-1, the tensor is partitioned into sub-tensors and decomposed independently and in phase-2, partial decomposition results are stitched through iterative block-centric refinement into a combined decomposition.

In Section 6, we recognized that in many applications, users may have a focus of interest – i.e., part of the data for which the user needs high accuracy– and beyond this area focus, the user may not be interested in maintaining high accuracy. Therefore, I proposed Personalized Tensor Decomposition (PTD) technique to maintain accuracies taking into account user feedback.

However, PTD cannot handle evolving user interest in an efficient way – i.e., when user interest evolves to different position, PTD requires re-decomposition based on user's new preference and this can have high computational cost and turn into a system bottleneck.

In PTD, user interest is modeled as a sub-tensor and when user's interest evolves to a different place, we can repartition the tensor based on the new interest boundary. One

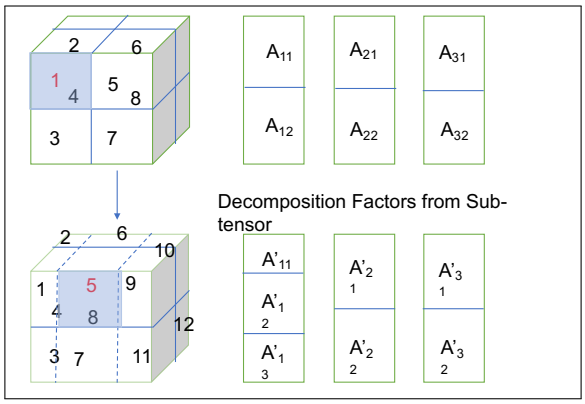


Figure 7.1: Dynamic Evolving User Focus Block: Left: Shaded Block Represent User Focus Which Evolves from Block-1 (with Old Partitions) to Block-5 (with New Partitions); Right: Sub-tensor Decomposition Factors with Old Partitions and New Partitions

interesting observation from 2PCP Algorithm 2's Phase-2 execution is that decomposition factors of old sub-tensor can still contribute to the overall tensor re-construction based on the new interest position. But the accuracy may suffer based on the information overlap between old interest block and the new interest block.

7.1.1 Contributions: Dynamic Evolving User Focus Block Tensor Decomposition (DPTD)

Relying on the observation, in this chapter, I propose Dynamic Evolving User Focus Block Tensor Decomposition (DPTD) which can intelligently reuse the existing decomposition result to improve evolving focus block decomposition efficiency.

Specifically dynamic evolving user focus, we propose to reuse previous decomposition results to avoid re-decomposition as much as possible and save execution time to improve decomposition efficiency. We propose four alternative strategies to improve the overall (time and accuracy) efficiency in dealing with preference evolution in tensor analysis

- RR: Repartition, Re-decomposition:
  - repartition the whole tensor based on the new interest boundary, build a connec-

tion graph, assign a decomposition rank to each sub-tensor based on PPR, and conduct sub-tensor decomposition

  - leverage two phase CP decomposition (2PCP) phase-2 refinement to obtain the final decomposition factor

- OR: Old partition, Re-decomposition:

  - reuse old partition schema, re-assign decomposition rank of sub-tensors based on the relationship graph with the new interest
  - re-decompose sub-tensors based on the new rank.
  - leverage two phase CP decomposition (2PCP) phase-2 refinement to obtain the final decomposition factor

- OP: Old partition, Partial Re-decomposition:

  - reuse old partition schema, re-assign decomposition ranks of sub-tensors based on the relationship graph with the new interest,
  - only redecompose sub-tensors with the new decomposition rank larger than the old one. For the cases where new rank is smaller or equal to the old rank, re-use old decomposition result and truncate the sub-factor based on the new rank
  - leverage two phase CP decomposition (2PCP) phase-2 refinement to obtain the final decomposition factor

- OP*: Old partition, Partial Re-decomposition, Optimized initialization

  - reuse old partition schema, re-assign decomposition ranks of sub-tensors based on the relationship graph with the new interest,
  - for the cases where new rank is smaller or equal to the old rank, re-use decomposition results (truncate the sub-factor)

– for the cases where the new rank is larger than the old rank, initiate the decomposition sub-factors with the old decomposition results plus random initialization

– leverage two phase CP decomposition (2PCP) phase-2 refinement to obtain the final decomposition factor

In Chapter 7.3, we compare these four strategies under different scenarios.

7.2 Dynamic Evolving User Focus Block Tensor Decomposition (DPTD)

The proposed algorithm is built upon two phase block based CP composition(2PCP) [34], further optimized for dynamic evolving user interest scenarios. The 2PCP algorithms consist of two phases:

- Phase-1: The given tensor is partitioned into blocks / sub-tensors, and each block is decomposed independently. To obtain sub-tensor decomposition ranks, we leverage PTD Sub-tensor Impact Graph (Chapter 3) and rely on personalized PageRank (PPR [6]) to measure sub-tensor relatedness.

- Phase-2: The partial decomposition results are stitched into a combined decomposition through an iterative block-centric refinement process

This work leverage phase-2 refinement, and the four strategies are implemented by revising phase-1 algorithm, shown in Algorithm 8.

7.2.1 RR: Repartition, Re-decomposition

RR-step 1: Repartition: In the Figure 7.2, blue shaded area represents the old interest and orange shaded area represents the new interest. Based on block-centric tensor decomposition, we re-partition the whole tensor based on the new user interest. In Figure 7.2, old interest is block-1 in the upper tensor and new interest is block-5 in the lower tensor.

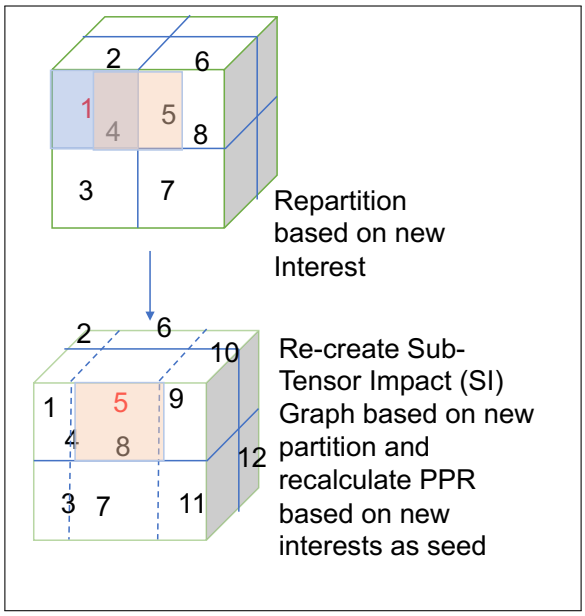


Figure 7.2: RR: Repartition, Re-decomposition

RR-step 2: Re-decomposition: For each sub-tensor, we conduct full decomposition with CPALS.

7.2.2 OR: Old partition, Re-decomposition

OR-step 1: Old partition: In this case, we keep the old partition schema but recalculate ranks. In Figure 7.3, new interest (orange shaded area) overlaps on block-1 and block-5, we mark block-1 and block-5 as new interested blocks. We use two new interest blocks as seeds to recompute PPR scores. $maxRank_{new}$ is then calculated as

$$maxRank_{new} = maxRank_{old} \times numberOfInterest_{old}/numberOfInterest_{new},$$

This helps ensure fair resource usage since the rank of decomposition will decide the resource consumed.

OR-step 2: Re-decomposition: For each sub-tensor, we conduct full decomposition with

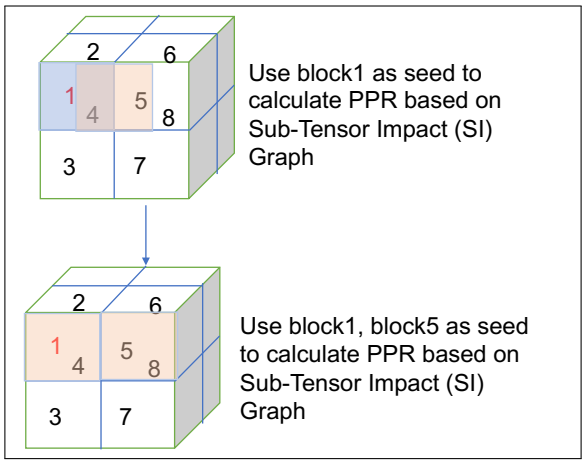
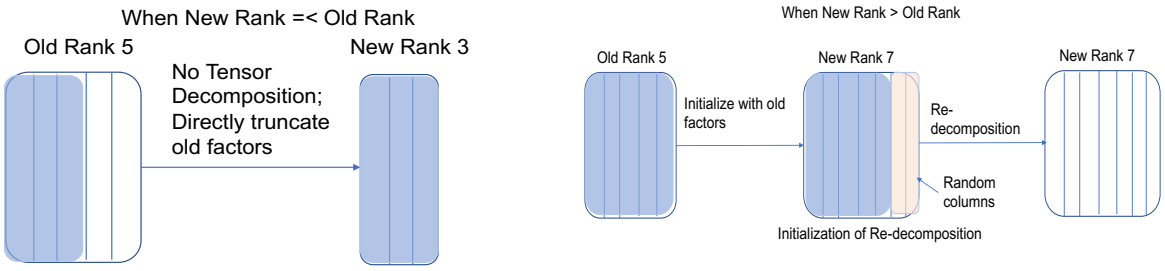


Figure 7.3: OR/OP/OP*: Old Partition, Re-decomposition with Different Optimization Strategies



(a) OP/OP*: When New Rank < Old Rank (b)OP*: When New Rank > Old Rank

Figure 7.4: Illustration of OP and OP* Re-decomposition Optimization

CPALS based on new ranks.

7.2.3 OP: Old partition, Partial Re-decomposition

OP-step 1: Old partition: Refer to OR-step 1.

OP-step 2: Partial Re-decomposition: We can avoid some computation by leveraging old decomposition result. Especially when $rank_{new} =< rank_{old}$, we truncate the previous sub-tensor decomposition factor by $rank_{new}$, assuming the factors encode critical information in decreasing manner from left column to right column. This is visualized in Figure 7.4.

Algorithm 8 Dynamic Evolving User Focus Block Tensor Decomposition (DPTD)

Input: original tensor, $\boldsymbol{\mathcal{X}}$; $\mathcal{T}_{old}$, the set of sub-tensors in old partition schema; Δ_{old}, the old update; Δ_{new}, the new update; U, sub-tensor decomposition factors based on old update

1. *Phase 1:*
   (a) If RR
      i. Repartition tensor based on Δ_{new}, get set of sub-tensors $\mathcal{T}_{new}$ based on new partition schema and $\mathring{\boldsymbol{\mathcal{X}}}_{\Delta_{new}}$ as new focus sub-tensor
      ii. Build the Sub-tensor Impact graph $\mathcal{D}_{new}$ based on $\mathcal{T}_{new}$
      iii. Calculate decomposition rank for each sub-tensor based on new partition schema $F_{\vec{d}} = F \times \frac{PPR(\mathring{\boldsymbol{\mathcal{X}}}_{\Delta_{new}})}{max\{PPR(\mathcal{D}_{new})\}}$
      iv. for all $\vec{t} \in \mathcal{T}_{new}$
         - Apply tensor decomposition CP-ALS for $\vec{t}$ based on $F_{\vec{d}}$
   (b) If OR, OP or OP*
      i. update new focus sub-tensors based on old partition schema: $\boldsymbol{\mathcal{X}}_{\delta_{new}}$;
      ii. Re-calculate decomposition rank for each sub-tensor based on new focus sub-tensor from old partition schema $F_{\vec{d}} = F \times \frac{PPR(\boldsymbol{\mathcal{X}}_{\delta_{new}})}{max\{PPR(\mathcal{D})\}}$
      iii. for all $\vec{t} \in \mathcal{T}_{old}$
         - If OR, apply tensor decomposition CP-ALS for $\vec{t}$ based on $F_{\vec{d}}$
         - If OP, apply tensor decomposition CP-ALS for $\vec{t}$ based on $F_{\vec{d}}$ when $F_{\vec{d}} > F_{\vec{d}_{old}}$; truncate $U_{\vec{t}}(1 : F_{\vec{d}})$ for decomposition result when $F_{\vec{d}} <= F_{\vec{d}_{old}}$,
         - If OP*,
            - when $F_{\vec{d}} > F_{\vec{d}_{old}}$, generate initialization factor with $U_{\vec{t}}$ and random assigment, apply tensor decomposition CP-ALS for $\vec{t}$ based on $F_{\vec{d}}$ and the initialization
            - when $F_{\vec{d}} <= F_{\vec{d}_{old}}$, truncate $U_{\vec{t}}(1 : F_{\vec{d}})$ for decomposition result
2. *Phase 2:* leverage 2PCP phase-2 refinement

7.2.4 OP: Old partition, Partial Re-decomposition, Optimized initialization*

OP*-step 1: Old partition: Refer to OR-step 1.

OP*-step 2: Partial Re-decomposition, Optimized initialization: Besides partial re-decomposition in OP, OP* further optimizes re-decomposition of sub-tensors whose $rank_{new} > rank_{old}$. In this case, we initialize with old factors and add random columns as indicated in Figure 7.4, we then conduct decomposition with CPALS based on the initialization factors. We expect that the decomposition process will converge faster given the initialization.

7.3 Experiments

In this section, we report experiment results to assess the efficiency and effectiveness of the four strategies under different scenarios.

7.3.1 Experiment Setup

Data Sets. In this section, we use both sparse and dense data sets. As dense data sets, we use real video datasets [62] with size: 480*720*426: which means that temporal mode size is 426 (426 frames) and each frame size is 480*720. As sparse data sets, we consider *Enron* [59] email data set and *MovieLens* [20] data set. The first data set has dimensions $5632 \times 184 \times 184$, density 1.8×10^{-4}, and schema, $\langle time, from, to \rangle$. Each value represents existence: if there is an email between from and to, the value is 1 otherwise 0. The *MovieLens* data set ($943 \times 1682 \times 2001$, density 3.15×10^{-5}) has schema $\langle user, movie, time \rangle$. Each value represents the rating value between 1 and 5 (or 0 if there is no rating).

Evaluation Scenarios. As mentioned in Section 7.1, accuracy performance can depend on information overlap between old and new interest regions. For example, if two regions have large overlaps, in this case, we may expect more information can be reused from the old result; However, when two regions have small overlap, we may expect that re-using old results can cause accuracy loss. Given this, we consider four scenarios (Figure 7.5) ranging from high overlap to no overlap. For each scenario, we randomly generate 10 to 30 cases; each case is run 30 times and we report average results. In detail, we describe the steps of generating the experiment user focus:

- step-1: get the input tensor and prepare two partition schema, and partition the input tensor into two set of sub-tensors: one is used for old interest, the other one is used for new interest. Build the mapping relationship between old and new partitions based on overlap area.

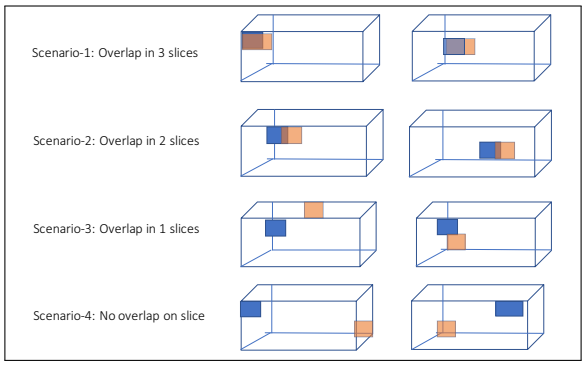


Figure 7.5: Four Overlap Scenarios Illustration: Blue Shaded and Orange Shaded Blocks Represent User's Old and New Interest, Four Scenario Is Illustrated Based on the Number of Overlap Slices

- step-2: randomly generate block number from each partition schema and found the number of common slice based on mapping relationship.
- step-3: decide scenario based on the number of common slices

Evaluation Criteria. We report accuracies for user's area of focus. We also report the decomposition time for different algorithms[1]. For all the experiments, we use max decomposition rank 10.

Decomposition Strategies. We consider four alternative strategies (RR/OR/OP/OP*) on sparse and dense datasets.

Hardware and Software. We used a quad-core Intel(R) Core(TM)i5-2400 CPU @ 3.10GHz machine with 16.00GB RAM. All codes were implemented in Matlab and run using Matlab 7.11.0 (2010b) and Tensor Toolbox Version 2.6. [3].

[1]Note that the time to compute rank assignments for the partitions is negligible (e.g. on the order of few milliseconds for $4 \times 4 \times 4$ partitioning of the tensor); therefore we do not report and investigate rank assignment times.

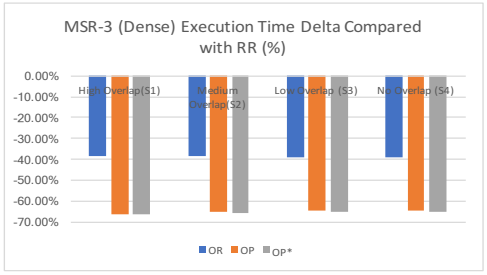
7.3.2 Discussion of the Results

In this section, we evaluated experiment result on execution times and accuracies of several proposed approaches in different scenarios. Here RR refers to the naive re-partition and re-decomposition strategy where we re-compute everything based on new interest. OR, refers to old partition, re-decomposition strategy in which we can leverage old partition schema, but re-decompose sub-tensors based on the new Sub-tensor Impact (SI) graph. OP refers to old partition, partial re-decomposition strategy in which we leverage old partition schema but execute partial re-decomposition only when new decomposition rank is larger than the old rank (we re-use old result when new rank is less than old rank). OP* refers to the old partition, partial re-decomposition with optimized initialization strategy in which we further optimize re-decomposition process initialization through old decomposition factors.

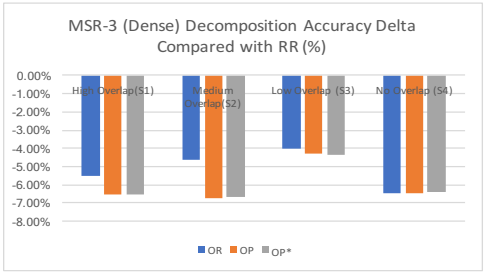
Initial Expectations

We expect RR and OR to have larger execution times since tensor re-decomposition takes more time. We also expect that to provide better accuracy. Especially RR, since re-decomposition is based on exact new interest region. OP is expected to have less execution time but lower accuracy than RR/OR since it re-uses old decomposition result. OP* is expected to be the fastest algorithm since it not only leverage partial decomposition, but also attempt to intelligent initialization.

Consider the scenarios in Figure 7.5, we expect higher accuracy for OR, OP and OP* in high overlap scenarios (e.g. S1 and S2) because more overlap means more useful information contained when reusing old decomposition results. Therefore, we expect small accuracy drop from RR and OR/OP/OP*.



(a) Execution Time Delta



(b)Decomposition Accuracy Delta

Figure 7.6: Experiment Results for Dense Dataset Msr Video: Delta Metrics with Comparison to RR: $OR\_time_{delta} = (OR\_time - RR\_time)/RR\_time$ And Negative Value Indicates Less Execution Time than RR

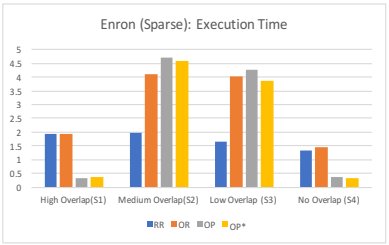
Results for the Dense Datasets

Figure 7.6 compares execution time and accuracy of the alternative approaches in different scenarios. As expected, execution time is $RR > OR > OP \approx OP^*$. Also as expected, RR has the highest accuracy. The second best candidate is OR. Meanwhile, OP and OP* has comparable accuracy. The least overlap scenario(S4) has the largest accuracy drop between RR and the other three algorithm.

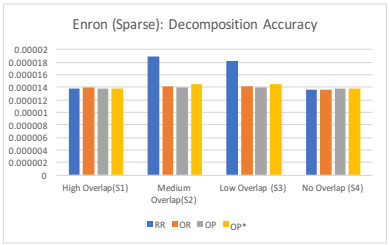
Results for Sparse Datasets

Figure 7.7 describe execution times and accuracies on sparse dataset (Enron and Movielens). Unexpectedly, RR has lesser execution times than OR, or even OP and OP* in some cases. Obviously sparse dataset result is surprising, and we observe diffculties on reusing old decomposition results.

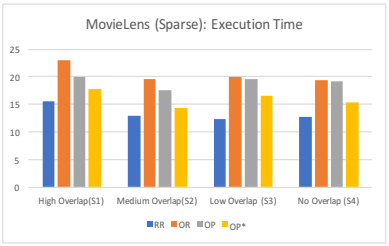
EXECUTION TIME **DECOMPOSITION ACCURACY**



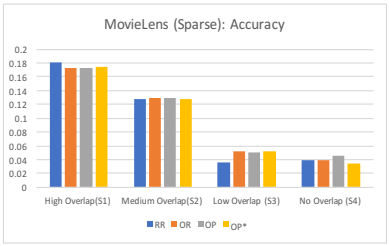
(a) Enron Execution Time



(b) Enron Decomposition Accuracy



(c) MovieLens Execution Time



(c) MovieLens Decomposition Accuracy

Figure 7.7: Experiment Results for Sparse Datasets: Enron and MovieLens

Summary

The above results indicate more confident usage in dense datasets. OR can save 39% execution time losing just around 5% accuracy compared with RR while OP and OP* can save up to 65% time losing only 6% accuracy. Therefore, DPTD is recommended assuming improve efficiency on dense data sets such as video pattern extraction. OP and OP* are recommended when we are sensitive on execution times, while RR should be considered only when we emphasize decomposition accuracy.

Chapter 8

CONCLUSION

The main goal of this dissertation is to optimize tensor decomposition for two major challenges - data focus and data evolution. Specifically, I inspected the two challenges with proposing several solutions to tackle each challenges under different scenario.

8.1 Sub-tensor Impact Graph

To deal with data explosion issue in tensor analytics, block-based tensor decomposition partition tensor into sub-tensors and execute in two phases: phase-1, sub-tensors are decomposed in parallel; phase-2, sub-tensors' factors are combined through iterative refinement process to compose final tensor decomposition factor. Sub-tensor impact graph is proposed to account for the inaccuracy propagation and impact score is used to present the sub-tensor decomposition impact on the others' decomposition. To properly account for the error propagation, I proposed four strategies on weight function which cover uniform weight, weight based on surface of interaction area, value distribution along the share mode, and combined method of interaction surface and value distribution. Furthermore, I shared two key observations which drive three optimization strategies that can improve efficiency of phase-2 refinement process which are applied in Chapter 5, Chapter 6.

8.2 Robust Personalized PageRank

Tensor can be partitioned into many sub-tensors,each sub-tensor have neighbors and connections with neighbors. This can form a network: each sub-tensor is a node, edge may be regarded as the connection between two sub-tensors. Therefore, we would like to capture the potential network topology to improve the block-based tensor decomposition

and measure the significance of the node to analyze the connections among nodes. we have shown that conventional personalized PageRank (PPR) algorithms associate *unnecessarily high bias* to the seed nodes and this negatively affects the node rankings when the seed set is *incomplete* and/or *noisy*. Therefore, *robust personalized PageRank* (RPR) algorithm is proposed to eliminate the potential noise in the seed set. We have shown that a novel *teleportation discounting* technique ensures that rankings are not overly biased towards the seed nodes and a novel *seed-set maximal PPR principle* helps differentiate among the seeds by considering the overall context defined by the seed set.

8.3 Block-Incremental CP Decomposition with Update Sensitive Refinement

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. Therefore, two-phase block-based incremental CP tensor decomposition (BICP) was introduced: 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.

8.4 Focusing Decomposition Accuracy by Personalizing Tensor

Due to the approximate nature of the tensors decomposition process, one way to reduce computational requirements might be to trade performance with accuracy. However, natu-

rally, a drop in accuracy may not be acceptable in many applications. Therefore, this is not a feasible solution to tackle the computational cost. In many applications, the user may have a focus of interest and beyond this area focus, the user may not be interested in maintaining high accuracy. Therefore, a novel *Personalized Tensor Decomposition (PTD)* mechanism was proposed for accounting for the user's focus and interests during tensor decomposition: PTD takes as input (a) an input tensor and (b) user's interest in the form of one or more area of focus. Given this input, PTD performs the tensor decomposition operation in such a way that, when reconstructed, the accuracy of the decomposition is boosted within the high-priority areas of focus. The proposed algorithm partitions the given tensor based on the focus of the user and assigns different initial decomposition ranks for different partitions. One of the key challenges we addressed in this paper is to account for the impact of the initial decompositions of the partitions on the final accuracies of the high-priority partitions in the user's focus. Experimental results showed that PTD is very effective in boosting accuracy for high priority regions of the tensor, while reducing the decomposition time.

8.5 Dynamic Evolving User Preferred Block Tensor Decomposition

Time series data has a special property that data will evolve over the time. For example, user interested data blocks may change as time evolve. Given the complexity of tensor decomposition, although PTD is efficient, re-computation for each user preference update can be bottleneck for the system. Therefore, dynamic evolving preferred tensor decomposition(DPTD) is proposed which can smartly reuse the existing decomposition result to improve evolving updated preferred block decomposition efficiency. The major challenge we addressed is to avoid full re-decomposition when user interest has evolving to a different position. Experiment results on dense dataset indicates that DPTD can significantly reduce execution time while maintaining decomposition accuracy within accepted loss.

Chapter 9

FUTURE WORK

9.1 Tucker Based Two-Phase Block-Centric Tensor Decomposition

Two-phase block based CP tensor decomposition [34] is proposed to tackle computational complexity challenges in tensor analytics and has been applied to several different scenario to optimize problems e.g data evolution, user focus and etc. Tucker has been one of the most popular tensor decomposition which are widely used in many application such as data dimensionality reduction, feature extraction, de-noising images or videos and etc. It would be much useful if we can extend the blocked-centric tensor decomposition framework with Tucker decomposition.

9.2 Tensor Analytic Optimization with Distributed Computation

One of another improvement extended from existing work is to allocate the optimization strategies proposed here to distributed systems. As mentioned in the work, we partition the tensors into small sub-tensors and apply different optimization strategies to tackle challenges existed in different domains and enable widely application. The property of partition and combining fits well with parallel computing and if we can further optimize the remote computation communication cost through the combining refinement process, this will bring much efficiency gain from tensor analytics.

9.3 Tensor Decomposition Application in Neural Network

There has been discussion on applying tensor decomposition to compress complicated neural network to maintain its expressive power [50]. To further improve the efficiency, we

can consider applying optimization strategies based on block-centric tensor decomposition to compress neural network which can deal with large-scale data, dynamically evolving data, focus of interest data, and shifting of focus data.

9.4 Efficient Temporal Pattern Extraction with Three-way DEDICOM Model

Three-way DEDICOM [32] is a higher-order extension of the DEDICOM model that incorporates a third mode of the data that is appropriate for problems that involve interactions between factors or when more extendibility than CP is needed but not as much as Tucker. Temporal data pattern extraction has been an important topic since there are more and more application focused on mining critical information from time series data. There are few research to apply three-way DEDICOM on time-series dataset such like video. It would be much beneficial to apply DEDICOM on the high-dimension time-series data to help analyze and recognize the hidden semantic of time-series data.

REFERENCES

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

[2] Avrachenkov K, et al. (2011) "Quick Detection of Top-k Personalized PageRank Lists". WAW, pages 50-61, 2011.

[3] Bader, B. W., T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6", Available online, URL `http://www.sandia.gov/ tgkolda/TensorToolbox/` (2015).

[4] B. W. Bader, R. A. Harshman, and T. G. Kolda, "Temporal Analysis of Semantic Graphs using ASALSAN", in ICDM 2007:Proceedings of the 7th IEEE International Conference on Data Mining, 2007, pp. 3342.

[5] Paul G. Brown. 2010. Overview of sciDB: large scale array storage, processing and analysis. InProceedings of the 2010 ACM SIGMOD International Conference on Management of data(SIGMOD '10). ACM, New York, NY, USA, 963-968.

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

[7] Bahmani B., et al. "Fast incremental and personalized PageRank". PVLDB. 4, 3, pages 173-184, 2010.

[8] B. Bahmani., K. Chakrabarti, D. Xin, "Fast personalized PageRank on MapReduce". In SIGMOD, pages 973-984, 2011.

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

[10] Borgatti MG., et al. "Network measures of social capital". Connections 21(2):27-36, 1998.

[11] Brin, S. and L. Page, "The anatomy of a large-scale hypertextual web search engine", Computer networks and ISDN systems **30**, 1-7, 107–117 (1998).

[12] Buckley C., Voorhees E.M. "Retrieval evaluation with incomplete information". SIGIR, pages 25-32, 2004.

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

[14] Chakrabarti S. "Dynamic personalized pagerank in entity-relation graphs". WWW, pages 571-580, 2007.

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

[16] Davidson, I., S. Gilpin, O. Carmichael and P. Walker, "Network discovery via constrained tensor analysis of fmri data", in "Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 194–202 (ACM, 2013).

[17] Fujiwara Y., et al. "Fast and exact top-k search for random walk with restart". PVLDB. 5, 5, pages 442-453, 2012.

[18] Z. Guan, et al. "Personalized tag recommendation using graph-based ranking on multi-type interrelated objects". SIGIR, 540-547, 2009

[19] Harshman, R. A., "Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis", (1970).

[20] Harper, F. M. and J. A. Konstan, "The movielens datasets: History and context", Acm transactions on interactive intelligent systems (tiis) **5**, 4, 19 (2016).

[21] R. A. Harshman, "Models for Analysis of Asymmetrical Relationships among n Objects or Stimul"i. In First Joint Meeting of the Psychometric Society and the Society for Mathemetical Psychology, McMaster University, Hamilton, Ontario, August 1978.

[22] Haveliwala T.H. "Topic-sensitive PageRank". WWW, 517-526, 2002.

[23] 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

[24] Jeon, I., E. E. Papalexakis, U. Kang and C. Faloutsos, "Haten2: Billion-scale tensor decompositions", in "Data Engineering (ICDE), 2015 IEEE 31st International Conference on", pp. 1047–1058 (IEEE, 2015).

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

[26] B. Kanagal, A. Ahmed, S. Pandey, V. Josifovski, L.G. Pueyo, and J. Yuan. "Focused matrix factorization for audience selection in display advertising". ICDE, 2013.

[27] H. Kim and H. Park, "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method", SIAM J. Matrix Anal. Appl., 30(2),pp. 713-730, 2008.

[28] Kim, M. and K. S. Candan, "Efficient static and dynamic in-database tensor decompositions on chunk-based array stores", in "Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management", pp. 969–978 (ACM, 2014).

[29] Kim, M. and K. S. Candan, "Tensordb: in-database tensor manipulation with tensor-relational query plans", in "Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management", pp. 2039–2041 (ACM, 2014).

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

[31] Kolda, T. G. and J. Sun, "Scalable tensor decompositions for multi-aspect data mining", in "Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on", pp. 363–372 (IEEE, 2008).

[32] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", SIAM Rev., vol. 51, no. 3, pp. 455500, August 2009.

[33] H. Lee, J. Yoo, and S. Choi, "Semi-supervised nonnegative matrix factorization", IEEE Signal Process. Lett., 2010.

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

[35] 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

[36] T. Maehara, T. Akiba, Y. Iwata, and K. Kawarabayashi. "Computing Personalized PageRank Quickly by Exploiting Graph Structures". VLDB14, pp. 1023-1034, 2014

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

[38] 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

[39] Phan, A. H. and A. Cichocki, "Parafac algorithms for large-scale problems", Neurocomputing **74**, 11, 1970–1984 (2011).

[40] Papalexakis, E. E., C. Faloutsos and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions", in "Joint European Conference on Machine Learning and Knowledge Discovery in Databases", pp. 521–536 (Springer, 2012).

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

[42] A.H. Phan and A. Cichocki. "Block decomposition for very large-scale nonnegative tensor factorization". CAMSAP, Workshop, 2009.

[43] Ioakeim Perros, Evangelos E Papalexakis, Fei Wang, Richard Vuduc, Elizabeth Searles, Michael Thompson, and Jimeng Sun. "Spartan: Scalable parafac2 for large & sparse data". arXiv preprint arXiv:1703.04219, 2017.

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

[45] 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)

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

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

[48] Y. Sun, J. Gao, X. Hong, B. Mishra, and B. Yin. "Heterogeneous tensor decomposition for clustering via manifold optimization", IEEE transactions on pattern analysis and machine intelligence, vol. 38, no. 3, pp. 476489, 2016.

[49] Emad Soroush, Magdalena Balazinska, and Daniel Wang. 2011. "ArrayStore: a storage manager for complex parallel array processing". InProceedings of the 2011 ACM SIGMOD International Conference on Management of data(SIGMOD '11). ACM, New York, NY, USA, 253-264.

[50] A.Tjandra, et al., "Tensor Decomposition for Compressing Recurrent Neural Network", IJCNN, 2018

[51] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. "Fast Random Walk with Restart and Its Applications". ICDM 06, pp. 613-622, 2006

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

[53] Malewicz G., et al. "Pregel: a system for large-scale graph processing". SIGMOD, pages 135-146, 2010.

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

[55] Y. Wang,Y. Hu, S. Kambhampati, B. Li, "Inferring Sentiment from Web Images with Joint Inference on Visual and Social Cues: A Regulated Matrix Factorization Approach". ICWSM, 2015

[56] Jibing Wu, Zhifei Wang, Yahui Wu, Lihua Liu, Su Deng, and Hongbin Huang, "A Tensor CP Decomposition Method for Clustering Heterogeneous Information Networks via Stochastic Gradient Descent Algorithms" Scientific Programming, vol. 2017, Article ID 2803091, 13 pages, 2017.

[57] White D.R., et al. "Betweenness centrality measures for directed graphs". Social Networks, 16, pages 335-346,1994.

[58] Q. Zhang, M. W. Berry, B. T. Lamb, and T. Samuel. "A parallel nonnegative tensor factorization algorithm for mining global climate data". InProceedings of the 9th International Conference on Computational Science, pages 405415, 2009.

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

[60] "http://www.jiliang.xyz/trust.html"

[61] "http://www.imdb.com/"

[62] https://www.cs.utexas.edu/ chaoyeh/web_action_data/dataset_list.html