IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

nTD: Noise-Profile Adaptive Tensor Decomposition.

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1647177 since 2017-08-29T01:41:59Z

Publisher:

International World Wide Web Conferences Steering Committee

Published version:

DOI:10.1145/3038912.3052641

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

07 January 2023

nTD: Noise-Profile Adaptive Tensor Decomposition *

Xinsheng Li
Arizona State University
Tempe, AZ, 85282, USA
lxinshen@asu.edu

K. Selçuk Candan
Ariona State University
Tempe, AZ, 85282, USA
candan@asu.edu

Maria Luisa Sapino
University of Torino
I-10149 Torino, Italy
marialuisa.sapino@unito.it

ABSTRACT

Tensor decomposition is used for many web and user data analysis operations from clustering, trend detection, anomaly detection, to correlation analysis. However, many of the tensor decomposition schemes are sensitive to noisy data, an inevitable problem in the real world that can lead to false conclusions. The problem is compounded by over-fitting when the user data is sparse. Recent research has shown that it is possible to avoid over-fitting by relying on probabilistic techniques. However, these have two major deficiencies: (a) firstly, they assume that all the data and intermediary results can fit in the main memory, and (b) they treat the entire tensor uniformly, ignoring potential non-uniformities in the noise distribution. In this paper, we propose a *Noise-Profile Adaptive Tensor Decomposition* (`nTD`) method, which aims to tackle both of these challenges. In particular, `nTD` leverages a grid-based two-phase decomposition strategy for two complementary purposes: firstly, the grid partitioning helps ensure that the memory footprint of the decomposition is kept low; secondly (and perhaps more importantly) any a priori knowledge about the noise profiles of the grid partitions enable us to develop a sample assignment strategy (or `s-strategy`) that best suits the noise distribution of the given tensor. Experiments show that `nTD`'s performance is significantly better than conventional CP decomposition techniques on noisy user data tensors.

1. INTRODUCTION

Tensors are commonly used for representing multi-dimensional data, such as user-centered document collections in the web and user interactions in social networks [20, 40]. [19], for example, incorporates contextual information to the traditional HITS algorithm, formulating the task as tensor decomposition. In [5], authors analyze the ENRON email social network using tensor decomposition and in [32], authors use tensors to incorporate user click information to improve web search.

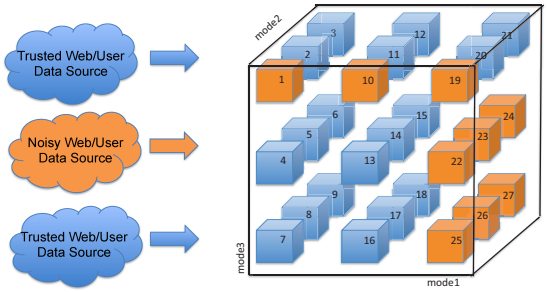


Figure 1: A sample 3-mode tensor, partitioned into a grid of sub-tensors, and its noise profile: the figure highlights (in orange) a subset of the sub-tensors which are noisy

Consequently, tensor decomposition operations (such as CP [13] and Tucker [34]) are increasingly being used to implement various data analysis tasks, from clustering, anomaly detection [20], correlation analysis [31], to pattern discovery [16]. Yet, tensor decomposition process is subject to several major challenges: One major challenge 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. Parallel implementations, such as GridParafac [26], GigaTensor [17], HaTen2 [15], TensorDB [18, 23, 22], were proposed to deal with the high computational cost of the task.

A second problem tensor decomposition faces is that the process can be negatively affected from noise and low quality in the data, which is especially a concern for web-based user data [6, 37, 38, 9] – in particular, especially for sparse data, avoiding over-fitting to the noisy data can be a significant challenge. Recent research has shown that it is possible to avoid such over-fitting by relying on probabilistic techniques [36], which introduces priors on the parameters, it can effectively average over various models and ease the pain of tuning parameters. Unfortunately, existing probabilistic approaches have two major deficiencies: (a) firstly, they assume that all the data and intermediary results can

*Research is supported by NSF#1318788 "Data Management for Real-Time Data Driven Epidemic Spread Simulations", NSF#1339835 "E-SDMS: Energy Simulation Data Management System Software", NSF#1610282 "DataStorm: A Data Enabled System for End-to-End Disaster Planning and Response", NSF#1633381 "BIGDATA: Discovering Context-Sensitive Impact in Complex Systems", and "FourCmodeling": EU-H2020 Marie Sklodowska-Curie grant agreement No 690817.

©2017 International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License.
WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3038914

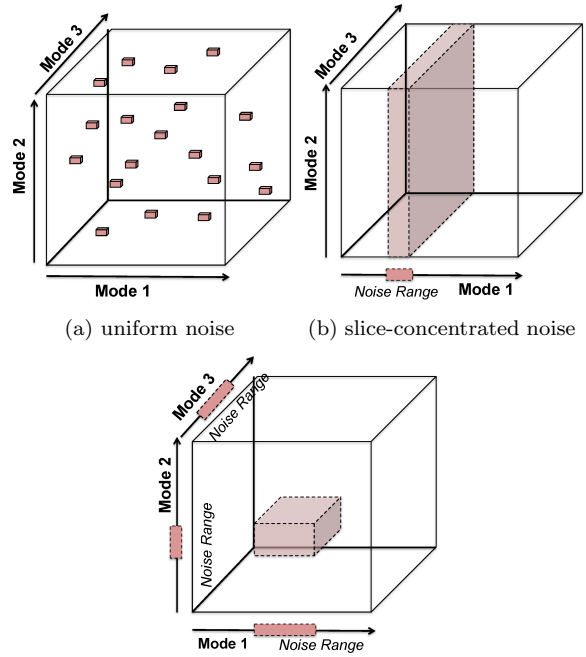
fit in the main memory and (b) they treat the entire tensor uniformly, ignoring possible non-uniformities in the distribution of noise in the given tensor.

In this paper, we propose a *Noise-Profile Adaptive Tensor Decomposition (*`nTD`*)* method, which leverages a priori information about noise in the data (which may be user provided or obtained through automated techniques [29, 12]) to improve decomposition accuracy. nTD partitions the user data tensor into multiple sub-tensors (Figure 1) and then decomposes each sub-tensor probabilistically through Bayesian factorization – the resulting decompositions are then recombined to obtain the decomposition for the whole tensor. Most importantly, `nTD` *provides a resource allocation strategy that accounts for the impact of the noise density of one sub-tensor on the decomposition accuracies of the other sub-tensors.* In other words, *a priori* knowledge about noise distribution among the sub-tensors (noise profiles depicted in Figures 1 and 2) is used to obtain a resource assignment strategy that best suits the noise distribution of the given tensor.

This paper is organized as follows: In the next section, we present the related work. Section 3 presents the relevant notations and the background. Section 4 describes the overview of the grid based probabilistic tensor decomposition scheme. Section 5 introduces the proposed sample assignment strategy (*s-strategy*) to adapt to different noise profiles, leading to a novel noise adaptive tensor decomposition (`nTD`) approach. Section 6 experimentally evaluates the effectiveness of the `nTD` and its alternative implementations. Experiments show that `nTD` indeed improves the decomposition accuracy of noise polluted tensors and the proposed sample assignment strategy (*s-strategy*) helps optimize the `nTD` performance under different noise scenarios. We conclude the paper in Section 7.

2. RELATED WORK

As we discussed in the previous section, tensor analysis is a commonly used technique for user-centered data analysis [28, 24, 1, 21, 35, 14]. Alternating least squares (ALS) is the most conventional method for tensor decomposition [13]: at each iteration, ALS estimates one factor matrix while maintaining other matrices fixed; this process is repeated for each factor matrix associated to the modes of the input tensor until convergence condition is reached. There are two widely used toolboxes for tensor manipulation: the *Tensor Toolbox for Matlab* [4] (for sparse tensors) and *N-way Toolbox for Matlab*[3] (for dense tensors). Yet, due to the significant cost [20] of tensor decompositions, various parallel algorithms and systems have been developed. [33] proposes MACH, a randomized algorithm that speedups the Tucker decomposition while providing accuracy guarantees. More recently, In [25], authors propose PARCUBE, a sampling based, parallel and sparsity promoting, approximate PARAFAC decomposition scheme. Scalability is achieved through sketching of the tensor (using biased sampling) and parallelization of the decomposition operations onto the resulting sketches. TensorDB [18, 23] leverages a block-based framework to store and retrieve data, extends array operations to tensor operation, and introduces optimization schemes for in-database tensor decomposition. HaTen2 [15] focuses on sparse tensors and presents a scalable tensor decomposition suite of methods for Tucker and PARAFAC decompositions on a MapReduce framework. SCOUT [16] is a recent coupled matrix-tensor factorization framework, also built on MapReduce. In addition to parallelism, it also leverages computation reordering as well as data transformation and reuse to reduce the computational cost of the process.



(a) uniform noise (b) slice-concentrated noise

(c) multi-modal noise

Figure 2: Alternative noise profiles of a tensor

In [11], authors develop a probabilistic framework, pTucker, for modeling structural dependency from partially observed multi-dimensional arrays. [39] implements a deterministic Bayesian inference algorithm, which formulates CP factorization with a hierarchical probabilistic model and employs Bayesian treatment by incorporating a sparsity-inducing prior over multiple latent factors and the appropriate hyperpriors over all hyperparameters, resulting in automatic rank determination. [27] proposed a Bayesian framework for low-rank decomposition of multiway missing observations tensor data. The method helps with the discovery of the decomposition rank from the data; moreover, inference scales linearly with the observation size, which helps the proposed approach scale very well. [10] proposes a loss function that helps the tensor decomposition process handle both Gaussian and grossly non-Gaussian perturbations.

3. BACKGROUND AND NOTATIONS

Intuitively, the tensor model maps a schema with N attributes to an N-modal array (where each potential tuple is a tensor cell). Tensor decomposition process generalizes the matrix decomposition process to tensors and rewrites the given tensor in the form of a set of *factor* matrices (one for each mode of the input tensor) and a core matrix (which, intuitively, describes the spectral structure of the given tensor). The two most popular tensor decomposition algorithms are the Tucker [34] and the CANDECOMP/PARAFAC (CP) [13] decompositions. While CP decomposes the input tensor into a sum of component rank-

one tensors (leading into a diagonal core), Tucker decomposition results in a dense core. In this paper, we focus on the CP decomposition process of user data tensors.

3.1 CP Decomposition

Given a tensor $\boldsymbol{\mathcal{X}}$, CP factorizes the tensor into factor matrices with F rows (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., $\mathbf{A} = [\, a_1 \; a_2 \cdots \; a_F \,]$. Since tensors may not always be exactly decomposed, the new tensor $\tilde{\boldsymbol{\mathcal{X}}}$ obtained by recomposing the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ is often different from the input tensor, $\boldsymbol{\mathcal{X}}$. The accuracy of the decomposition is often measured by considering the Frobenius norm of the difference tensor.

3.2 Parameters of Tensor Noise Profile

Noise distribution: Noise can be distributed in a tensor in several ways:

- In *uniform (uni) noise* (Figure 2(a)), there is no underlying pattern and noise is not clustered across any slice or region of the tensor.
- *Slice-concentrated (sc)* noise (Figure 2(b)) is clustered on one or more slices on the tensor across one or more modes. For example, a particular data source (represented by one or more slices) may be known to provide low quality, untrusted information.
- In *multi-modal (mm)* noise, again, the noise is clustered; however, in this case the noise is expected to occur when a combination of a subset of the values across two or more modes are considered together as in Figure 2(c).

Noise density: This is the ratio of the cells that are subject to noise. In this paper, without loss of generality, we assume noise is on cells that have values (i.e., the observed values can be faulty, but there are no spurious observations) and, thus, we define noise density as *a ratio of the non-null cells.*
Dependent vs. independent noise: Noise may impact the observed values in the tensor in different ways: in *value-independent noise*, the correct data may be overwritten by a completely random new value, whereas in *value-correlated noise* existing values may be perturbed (often with a Gaussian noise, defined by a standard deviation, σ).

4. GRID BASED PROBABILISTIC TENSOR DECOMPOSITION (GPTD)

As we described above, noise may not be uniformly distributed on a tensor. In order to take into account the underlying non-uniformities, we propose to partition the tensor into a grid and treat each grid partition differently based on its noise profile. In this section, we present a

Algorithm 1 Phase 1: Monte Carlo based Bayesian decomposition of each sub-tensor(extension of [36] to more than 3 modes)

Input: Sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, sampling number L
Output: Decomposed factors $\boldsymbol{U}^{(1)}_{\vec{k}}$, $\boldsymbol{U}^{(2)}_{\vec{k}}$, ..., $\boldsymbol{U}^{(N)}_{\vec{k}}$

1. Initialize model parameters $\boldsymbol{U}^{(1)1}_{\vec{k}}$, $\boldsymbol{U}^{(2)1}_{\vec{k}}$, ..., $\boldsymbol{U}^{(N)1}_{\vec{k}}$.
2. For $l = 1, \ldots, L$
   (a) Sample the hyper-parameter, α:
   - $\alpha^l \sim p(\alpha^l | \boldsymbol{U}^{(1)l}_{\vec{k}}, \boldsymbol{U}^{(2)l}_{\vec{k}}, \ldots, \boldsymbol{U}^{(N)l}_{\vec{k}}, \boldsymbol{\mathcal{X}}_{\vec{k}})$

   (b) For each mode $j = 1, \ldots, N$,
   i. Sample the corresponding hyper-parameter, Θ:
   - $\Theta_{\boldsymbol{U}^{(j)l}_{\vec{k}}} \sim p(\Theta_{\boldsymbol{U}^{(j)l}_{\vec{k}}} | \boldsymbol{U}^{(j)l}_{\vec{k}})$

   ii. For $i_j = 1, ..., I_j$, sample the mode (in parallel):

$$\boldsymbol{U}^{(j)(l+1)}_{\vec{k}(i_j)} \sim p\Big(\boldsymbol{U}^{(j)}_{\vec{k}(i_j)} \Big| \boldsymbol{U}^{(1)l}_{\vec{k}}, \ldots, \boldsymbol{U}^{(j-1)l}_{\vec{k}}, \boldsymbol{U}^{(j+1)l}_{\vec{k}}, \ldots, \boldsymbol{U}^{(N)l}_{\vec{k}}, \Theta^l_{\boldsymbol{U}^{(j)}_{\vec{k}}}, \alpha^l, \boldsymbol{\mathcal{X}}_{\vec{k}}\Big)$$

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

Grid Based Probabilistic Tensor Decomposition (GPTD) approach which extends the wholistic Probabilistic Tensor Decomposition (PTD [36]) into a grid-based framework. Note that, in and of itself, GPTD does not leverage **a priori** knowledge about noise distribution, but as we see in Section 5, it provides a framework in which noise-profile based adaptation can be implemented.

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., $|\mathcal{K}| = \prod_{i=1}^{N} K_i$. Given a target decomposition rank, F, the first step of the proposed decomposition (`GPTD`) scheme is to decompose each sub-tensor in $\mathfrak{X}$ with target rank F, such that for each $\boldsymbol{\mathcal{X}}_{\vec{k}}$, we have $\boldsymbol{\mathcal{X}}_{\vec{k}} \approx \boldsymbol{I} \times_1 \boldsymbol{U}^{(1)}_{\vec{k}} \times_2 \boldsymbol{U}^{(2)}_{\vec{k}} \cdots \times_N \boldsymbol{U}^{(N)}_{\vec{k}}$, where $\mathfrak{U}^{(i)} = \{\boldsymbol{U}^{(i)}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ denotes the set of F-rank sub-factors[1] corresponding to the sub-tensors in $\mathfrak{X}$ along mode i and $\boldsymbol{I}$ is the N-mode $F \times F \times \ldots \times F$ identity tensor, where the diagonal entries are all 1s and the rest are all 0s. Intuitively, given a sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$, each entry $\boldsymbol{\mathcal{X}}_{\vec{k}(i_1,i_2,i_3,\ldots,i_N)}$ can be expressed as the inner-product of N F-dimensional vectors: $\boldsymbol{\mathcal{X}}_{\vec{k}(i_1,i_2,\ldots,i_N)} \approx [\boldsymbol{U}^{(1)}_{\vec{k}(i_1)}, \boldsymbol{U}^{(2)}_{\vec{k}(i_2)} \ldots, \boldsymbol{U}^{(N)}_{\vec{k}(i_N)}]$. We discuss the sub-tensor decomposition process next.

4.1 Phase 1: Monte Carlo based Bayesian Decomposition of Sub-tensors

For decomposing individual sub-tensors, we rely on the probabilistic approach proposed in [30, 36]: i.e., we describe

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

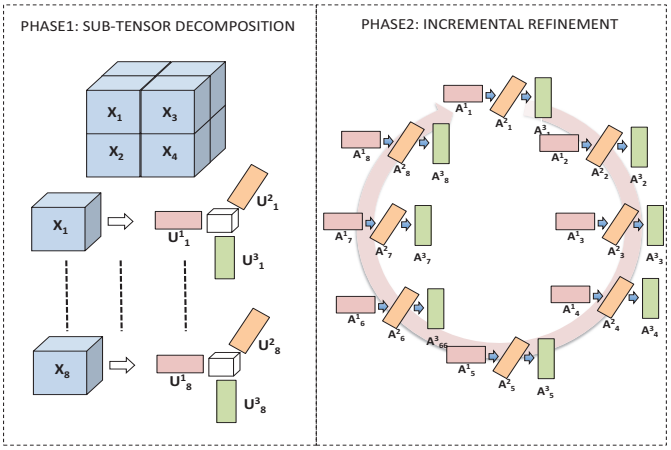


Figure 3: Illustration of sub-tensor based tensor decomposition: the input tensor is partitioned into smaller blocks, each block is decomposed (potentially in parallel), and the partial decompositions are stitched together through an iterative improvement process

the fit between the observed data and the predicted latent factor matrices, probabilistically, as follows:

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

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

$$\boldsymbol{U}^{(j)}_{\vec{k}(i_j)} \sim \boldsymbol{\mathcal{N}}(\mu_{\boldsymbol{U}^{(j)}_{\vec{k}}}, \Lambda^{-1}_{\boldsymbol{U}^{(j)}_{\vec{k}}}) \quad i_j = 1 \ldots I_j \tag{2}$$

where I_j is the dimensionality of the jth mode. Given this, one can estimate the latent features $\boldsymbol{U}^{(j)}_{\vec{k}}$ by maximizing the logarithm of the posterior distribution, $\log p(\boldsymbol{U}^{(1)}_{\vec{k}}, \boldsymbol{U}^{(2)}_{\vec{k}} \ldots, \boldsymbol{U}^{(N)}_{\vec{k}} | \boldsymbol{\mathcal{X}}_{\vec{k}})$. One difficulty with this approach, however, is the tuning of the hyper-parameters of the model: α and $\Theta_{\boldsymbol{U}^{(j)}_{\vec{k}}} \equiv \{\mu_{\boldsymbol{U}^{(j)}_{\vec{k}}}, \Lambda_{\boldsymbol{U}^{(j)}_{\vec{k}}}\}$ for $1 \leq j \leq N$. [36] notes that one can avoid the difficulty underlying the estimation of these parameters through a fully Bayesian approach, complemented with a sampling-based Markov Chain Monte Carlo (MCMC) method to address the lack of the analytical solution. The process is visualized in Algorithm 1 in pseudo-code form.

4.2 Phase 2: Iterative Refinement

Once the individual sub-tensors are decomposed, the next step is to stitch the resulting sub-factors into the full F-rank factors, $\boldsymbol{A}^{(i)}$ (each one along one mode), for the input tensor, $\boldsymbol{\mathcal{X}}$. Let us partition each factor $\boldsymbol{A}^{(i)}$ into K_i parts corresponding to the block boundaries along mode i:

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

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

$$\boldsymbol{\mathcal{X}}_{\vec{k}} \approx \boldsymbol{I} \times_1 \boldsymbol{A}^{(1)}_{(k_1)} \times_2 \boldsymbol{A}^{(2)}_{(k_2)} \cdots \times_N \boldsymbol{A}^{(N)}_{(k_N)} \tag{3}$$

The current estimate of the sub-factor $\boldsymbol{A}^{(i)}_{(k_i)}$ can be revised using the following update rule [26]:

$$\boldsymbol{A}^{(i)}_{(k_i)} \longleftarrow \quad \boldsymbol{T}^{(i)}_{(k_i)} \left(\boldsymbol{S}^{(i)}_{(k_i)}\right)^{-1} \tag{4}$$

where

$$\boldsymbol{T}^{(i)}_{(k_i)} = \sum_{\vec{m} \in \{[*,\ldots,*,k_i,*,\ldots,*]\}} \boldsymbol{U}^{(i)}_{\vec{m}} \left(\boldsymbol{P}_{\vec{m}} \oslash (\boldsymbol{U}^{(i)T}_{\vec{m}} \boldsymbol{A}^{(i)}_{(k_i)})\right)$$

$$\boldsymbol{S}^{(i)}_{(k_i)} = \sum_{\vec{m} \in \{[*,\ldots,*,k_i,*,\ldots,*]\}} \boldsymbol{Q}_{\vec{m}} \oslash \left(\boldsymbol{A}^{(i)T}_{(k_i)} \boldsymbol{A}^{(i)}_{(k_i)}\right)$$

such that, given $\vec{m} = [m_1, m_2, \ldots, m_N]$, we have

- $\boldsymbol{P}_{\vec{m}} = \circledast_{h=1}^{N} (\boldsymbol{U}^{(h)T}_{\vec{m}} \boldsymbol{A}^{(h)}_{(m_h)})$ and
- $\boldsymbol{Q}_{\vec{m}} = \circledast_{h=1}^{N} (\boldsymbol{A}^{(h)T}_{(m_h)} \boldsymbol{A}^{(h)}_{(m_h)})$.

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

4.3 Overview of GPTD

The two phases of the decomposition process are visualized in Algorithm 2 and Figure 3.

Algorithm 2 The outline of the GPTD process

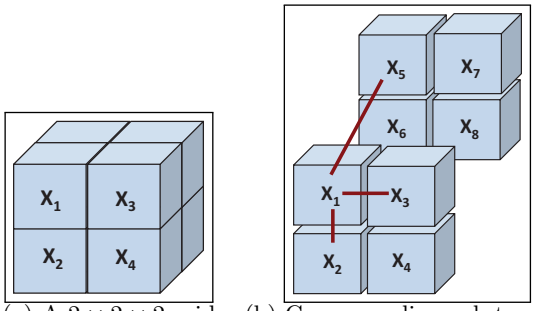
Input: Input tensor $\boldsymbol{\mathcal{X}}$, partitioning pattern $\mathcal{K}$, and decomposition rank, F, and per sub-tensor sample count, L

Output: Tensor decomposition $\mathring{\boldsymbol{\mathcal{X}}}$

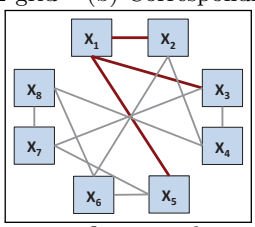
1. Phase 1: for all $\vec{k} \in \mathcal{K}$
   - decompose $\boldsymbol{\mathcal{X}}_{\vec{k}}$ into $\boldsymbol{U}^{(1)}_{\vec{k}}, \boldsymbol{U}^{(2)}_{\vec{k}}, \ldots, \boldsymbol{U}^{(N)}_{\vec{k}}$

   with sample count L using Algorithm 1.
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 $\mathring{\boldsymbol{\mathcal{X}}}$



(a) A $2 \times 2 \times 2$ grid (b) Corresponding sub-tensors



(c) Pairwise refinement dependencies

Figure 4: A sample grid and the corresponding pairwise refinement dependencies among the sub-tensors per Equation 4

5. NOISE-PROFILE ADAPTIVE TENSOR DECOMPOSITION

One crucial piece of information that the basic grid based decomposition process fails to account for is *potentially available knowledge* about the distribution of the noise across the input tensor. Note that, in the second phase of the process, each $\boldsymbol{A}^{(i)}_{(k_i)}$ is maintained incrementally by using, for all $1 \leq j \leq N$, (a) the current estimates for $\boldsymbol{A}^{(j)}_{(k_j)}$ and (b) 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. This implies that a sub-tensor which is poorly decomposed due to noise may negatively impact decomposition accuracies also for other parts of the tensor. Consequently, it is important to properly allocate resources to prevent a few noisy sub-tensors among all from negatively impacting the overall accuracy.

In [22], we studied how to allocate resources, in a way that takes into account, user's non-uniform accuracy preferences for different parts of the tensor. In this paper, we develop a novel noise-profile adaptive tensor decomposition (`nTD`) scheme that focuses on resource allocation based on noise distribution. More specifically, user provided or auomatically discovered [2, 37, 38] a priori knowledge about the noise profiles of the grid partitions enables us to develop a sample assignment strategy (or `s-strategy`) that best suits the noise distribution in a given tensor. In particular, `nTD` assigns the ranks and samples to different sub-tensors in a way that maximizes the overall decomposition accuracy of the whole tensor without negatively impacting the efficiency of the decomposition process. Since probabilistic decomposition can be costly, `nTD` considers *a priori* knowledge about each sub-tensor's noise density to decide the appropriate number of Gibbs samples to achieve good accuracy with the given number of samples.

5.1 Noise Sensitive Sample Assignment: First Naive Attempt

As we experimentally show in Section 6, there is a direct relationship between the amount of noise a (sub-)tensor has and the number of Gibbs samples it requires for accurate decomposition. On the other hand, the number of samples also directly impacts the cost of the probabilistic decomposition process. Consequently, given a set of sub-tensors, with different amounts of noise, uniform assignment of the number of samples, $L = \left(\frac{L_{(total)}}{|\mathcal{K}|}\right)$, where $L_{(total)}$ is the total number of samples for the whole tensor and $|\mathcal{K}|$ is the number of sub-tensors, may not be the best choice.

In fact, the numbers of Gibbs samples allocated to different sub-tensors $\boldsymbol{\mathcal{X}}_{\vec{k}}$ in Algorithm 1 *do not need to be the same.* As we have seen in Section 4.1, Phase 1 decomposition of each sub-tensor is independent from the others and, thus, the number of Gibbs samples of different sub-tensors can be different. This observation, along with observation that more samples can provide better accuracy for noisy sub-tensors, can be used to improve the overall decomposition accuracy for a given number of Gibbs samples. More specifically, the number of samples a noisy sub-tensor, $\boldsymbol{\mathcal{X}}_{\vec{k}}$, is allocated should be proportional to the density, $nd_{\vec{k}}$, of noise it contains:

$$L(\boldsymbol{\mathcal{X}}_{\vec{k}}) = \lceil \gamma \times nd_{\vec{k}} \rceil + L_{min}, \tag{5}$$

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

$$L_{(total)} = \sum_{\vec{k} \in \mathcal{K}} L(\boldsymbol{\mathcal{X}}_{\vec{k}}). \tag{6}$$

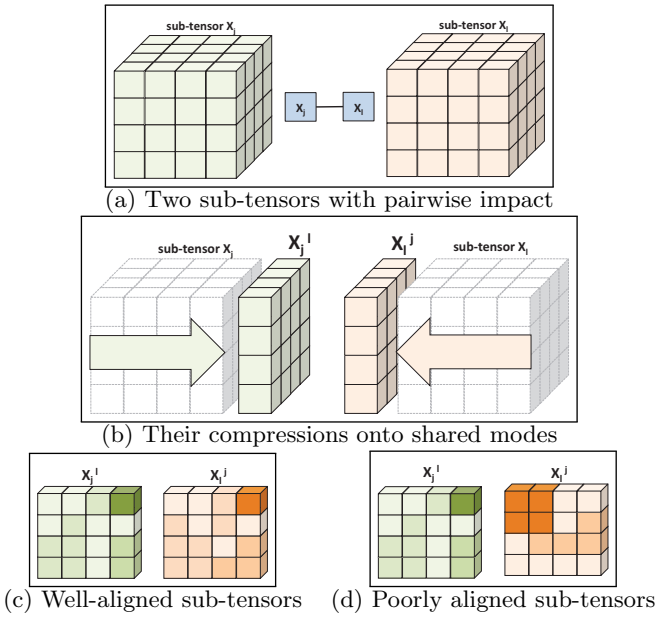
5.2 Noise Sensitive Sample Assignment: Second Naive Attempt

Equations 5 and 6, above, help allocate samples across sub-tensors based on their noise densities. However, they ignore the relationships among the sub-tensors. In Section 4.2, we have seen that, during the iterative refinement process of Phase 2, inaccuracies in decomposition of one sub-tensor can propagate across the rest of the sub-tensors. Therefore, a better approach could be to consider how errors can propagate across sub-tensors when allocating samples.

5.2.1 *Accounting for Accuracy Inter-dependencies among Sub-Tensors*

More specifically, in this section, we note that if we could assign a significance score to each sub-tensor, $\boldsymbol{\mathcal{X}}_{\vec{k}}$, that takes into account not only its noise density, but also the position of the sub-tensor relative to other sub-tensors, we could use this information to allocate samples.

Let $\boldsymbol{\mathcal{X}}$ be a tensor partitioned into a set (or grid) of sub-tensors $\mathfrak{X} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$. According to the update rule (Equation 4) in Section 4.2, if two sub-tensors are lined up along one of the modes of the tensor, they can be used to revise each other's estimates. This means that the update rule ties each sub-tensor's accuracy directly to $\sum_{1 \leq i \leq N} K_i$ other sub-tensors (that line up with the given sub-tensor along one of the N modes – see Figure 4).



(a) Two sub-tensors with pairwise impact

(b) Their compressions onto shared modes

(c) Well-aligned sub-tensors (d) Poorly aligned sub-tensors

Figure 5: Measuring the alignment of two sub-tensors: (a) the sub-tensors with pairwise impact, (b) their compressions onto their shared modes, (c) well-aligned tensors have similar distributions on this compressed representation, whereas (d) poorly aligned tensors have dissimilar distributions

Moreover, we see that 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, given two sub-tensors $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$, we can compute an *alignment* score, $align(\boldsymbol{\mathcal{X}}_{\vec{j}}, \boldsymbol{\mathcal{X}}_{\vec{l}})$, between $\boldsymbol{\mathcal{X}}_{\vec{j}}$ and $\boldsymbol{\mathcal{X}}_{\vec{l}}$ as $align(\boldsymbol{\mathcal{X}}_{\vec{j}}, \boldsymbol{\mathcal{X}}_{\vec{l}}) = cos(\boldsymbol{\mathcal{X}}_{\vec{j}}^{\vec{l}}, \boldsymbol{\mathcal{X}}_{\vec{l}}^{\vec{j}})$, where $cos()$ is the cosine similarity function and $\boldsymbol{\mathcal{X}}_{\vec{a}}^{\vec{b}}$ is the version of the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{a}}$ compressed, using the standard Frobenius norm, onto the modes along which the sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{a}}$ and $\boldsymbol{\mathcal{X}}_{\vec{b}}$ are aligned (Figure 5). Intuitively, this pairwise alignment score describes how the decomposition of one sub-tensor will impact another and also indicate the degree of numeric error propagation. A sub-tensor which is not aligned with the other sub-tensors is likely to have minimal impact on the accuracy of the overall decomposition even if it contains significant amount of noise. In contrast, a sub-tensor which is well-aligned with a larger portion of other sub-tensors may have a large impact on the other sub-tensors, and consequently, on the whole tensor. Consequently, while the former sub-tensor may not deserve a significant amount of resources, the accuracy of the latter sub-tensor is critical and hence that tensor should be allocated more resources to ensure better overall accuracy.

5.2.2 Sub-Tensor Centrality based Sample Assignment

Therefore, given pairwise alignment scores among the sub-tensors, one option is to measure the significance of a sub-tensor relative to other sub-tensors using a centrality measure like PageRank (PR [7]), which computes the significance of each node in a (weighted) graph relative to the other nodes. More specifically, given a graph, $G(V, E)$, the PageRank score $\vec{p}[i]$, of a node $v_i \in V$ is obtained by solving $\vec{p} = (1-\beta)\mathbf{A}\ \vec{p} + \beta\vec{s}$, where $\mathbf{A}$ denotes the transition matrix, β is a parameter controlling the random walk likelihood , and $\vec{s}$ is a teleportation vector such that for $v_j \in V$, $\vec{s}[j] = \frac{1}{\|V\|}$. Therefore, given (a) the set (or grid) of sub-tensors $\boldsymbol{\mathfrak{X}} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ and (b) their pairwise alignment scores, we can associate a significance score,

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

to each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$ by computing PageRank scores described by the vector $\vec{p}$. Given this score, we can then rewrite Equation 5 as

$$L(\boldsymbol{\mathcal{X}}_{\vec{k}}) = \lceil \gamma \times \tau_{\vec{k}} \times nd_{\vec{k}} \rceil + L_{min}, \tag{7}$$

taking into account both the noise density of the sub-tensor along with its relationship to other sub-tensors.

5.3 S-Strategy for Sample Assignment

The above formulation considers the position of each sub-tensor in the whole sub-tensor to compute its significance and then multiplies this with the corresponding noise density to decide how much resources to allocate to that sub-tensor. This, however, may not properly take into account the relationship among the noisy sub-tensors and the positioning of sub-tensors relative to the noisy ones.

In this paper, we note that a better approach would be to consider the noise densities of the sub-tensors directly when evaluating the significance of each sub-tensor. More specifically, instead of relying on PageRank, we propose to use a measure like personalized PageRank (PPR [8]), which computes the significance of each node in a (weighted) graph relative to a given set of seed nodes. Given a graph, $G(V, E)$, and a set, $S \subseteq G(V, E)$, of seed nodes, the PPR score $\vec{p}[i]$, of a node $v_i \in G(V, E)$ is obtained by solving $\vec{p} = (1-\beta)\mathbf{A}\ \vec{p} + \beta\vec{s}$, where $\mathbf{A}$ denotes the transition matrix, β is a parameter controlling the overall importance of the seeds, and $\vec{s}$ is a seeding vector such that if $v_i \in S$, then $\vec{s}[i] = \frac{1}{\|S\|}$ and $\vec{s}[i] = 0$, otherwise. Therefore, given (a) the set (or grid) of sub-tensors $\boldsymbol{\mathfrak{X}} = \{\boldsymbol{\mathcal{X}}_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$, (b) their pairwise alignment scores, and (c) a seeding vector

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

we associate a noise sensitive significance score,

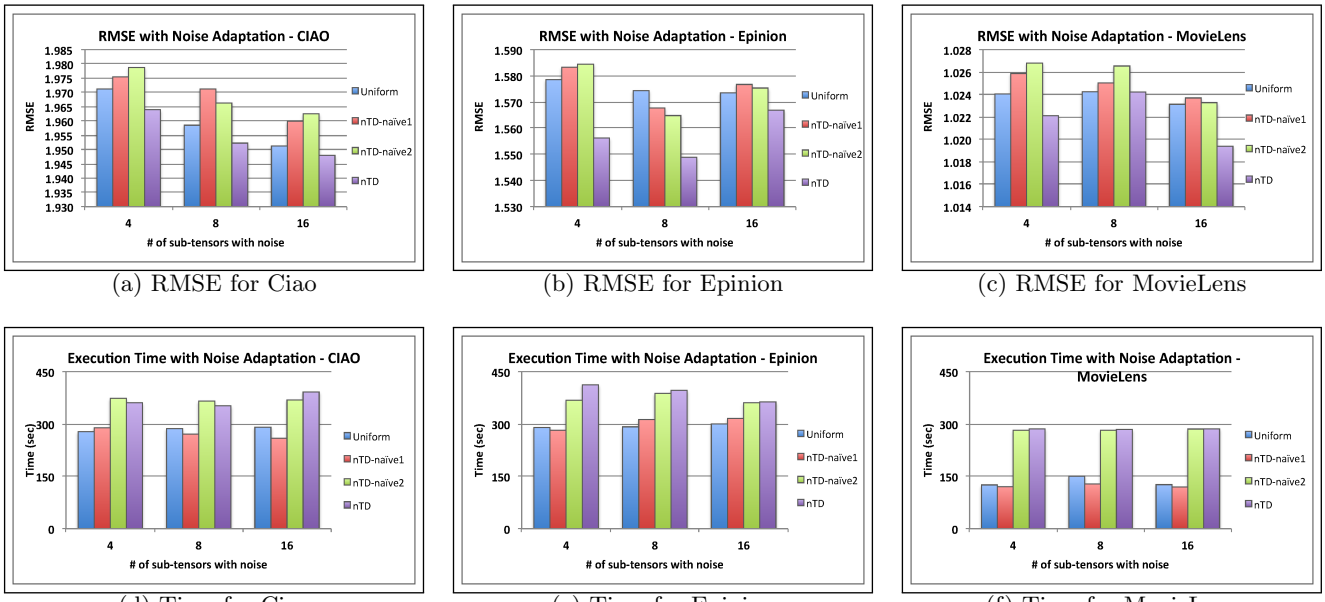
$$\eta_{\vec{k}} = \frac{\vec{p}[\vec{k}] - min_{\vec{j}\in\mathcal{K}}(\vec{p}[\vec{j}])}{max_{\vec{j}\in\mathcal{K}}(\vec{p}[\vec{j}]) - min_{\vec{j}\in\mathcal{K}}(\vec{p}[\vec{j}])},$$

to each sub-tensor $\boldsymbol{\mathcal{X}}_{\vec{k}}$ based on the PPR scores, described by the vector $\vec{p}$, relative to the noisy tensors. Given this score, we rewrite Equation 5 as

$$L(\boldsymbol{\mathcal{X}}_{\vec{k}}) = \lceil \gamma \times \eta_{\vec{k}} \rceil + L_{min}. \tag{8}$$

5.4 Overview of nTD

The pseudo-code of the proposed noise adaptive tensor decomposition (`nTD`) process is visualized in Algorithm 3.



(a) RMSE for Ciao (b) RMSE for Epinion (c) RMSE for MovieLens

(d) Time for Ciao (e) Time for Epinion (f) Time for MovieLens

Figure 6: RMSE and execution time (without sub-tensor parallelism) for `nTD` with different num. of noisy sub-tensors ($4 \times 4 \times 4$ grid; uniform noise; value independent noise; noise density 10%; total num. of samples = 640; $L_{min} = 9$, $F = 10$; max. num. of P2 iteration = 1000)

Algorithm 3 Overview of `nTD`: noise adaptive decomposition (with noise based resource allocation)

Input: original tensor $\boldsymbol{\mathcal{X}}$, partitioning pattern $\mathcal{K}$, noisy sub-tensor $\mathcal{K}_P$, and decomposition rank, F and total sampling number L

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

1. obtain the noise profile of the sub-tensors of $\boldsymbol{\mathcal{X}}$,
2. for sub-tensor $\vec{k} \in \mathcal{K}$, assign a decomposition rank $F_{\vec{k}} = F$ and a sampling number $L_{\vec{k}}$ based on noise-sensitive sample allocation strategy, described in Section 5.3.
3. obtain the decomposition, $\hat{\boldsymbol{\mathcal{X}}}$, of $\boldsymbol{\mathcal{X}}$ using the GPTD algorithm (Algorithm 2), given partitioning pattern $\mathcal{K}$ and the initial decomposition ranks $\{F_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$ and sampling number $\{L_{\vec{k}} \mid \vec{k} \in \mathcal{K}\}$,
4. Return $\hat{\boldsymbol{\mathcal{X}}}$

6. EXPERIMENTAL EVALUATION

In this section, we report experiments that aim to assess the effectiveness of the proposed *noise adaptive tensor decomposition* approach. In particular, we compare the proposed approach against another grid based strategy, Grid-Parafac. We further assess the proposed noise-sensitive sample assignment strategy (`s-strategy`) by comparing the performance of `nTD`, which leverages this strategy, against `GPTD` with uniform sample assignment, on user-centered data.

6.1 Experiment Setup

Key parameters and their values are reported in Table 1.

Data Sets. In these experiments, we used three user centered datasets: *Epinions* [41], *Ciao* [41], and *MovieLens* [**?**, 40]. The first two of these are comparable in terms of their sizes and semantics: they are represented in the form of $5000 \times 5000 \times 999$ (density 1.4×10^{-6}) and $5000 \times 5000 \times 996$ (density 1.7×10^{-6}) tensors, respectively, and both have the schema $\langle user, item, time \rangle$. The *MovieLens* data set ($943 \times 1682 \times 2001$, density 3.15×10^{-5}) is denser and has a different schema, $\langle user, movie, time \rangle$. In all three data sets, the tensor cells contain rating values between 1 and 5 or (if the rating does not exist) a special "null" symbol.

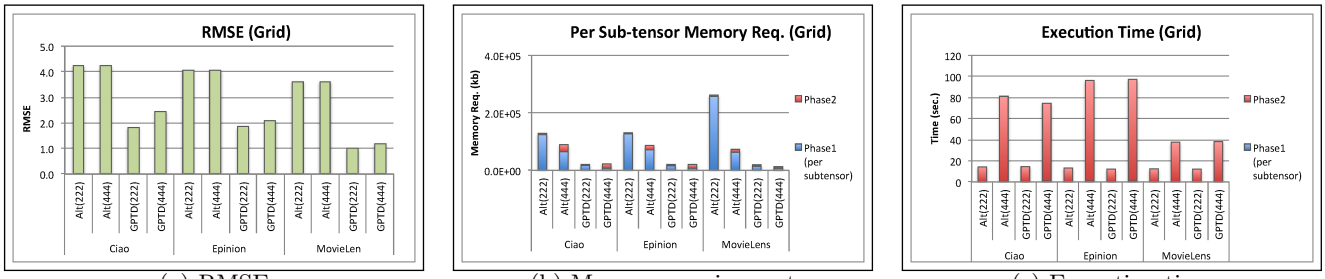
Parameters	*Alternative values*
Noise Density	**10**%; 30%; 50%; 80%
# partitions	$2 \times 2 \times 2$; $\mathbf{4 \times 4 \times 4}$
Per sub-tensor Gibbs sample count	1; **3**; 5; 10; 30; 80
Target Rank (F)	**10**

Table 1: Parameters – default values, used unless otherwise specified, are highlighted

Noise. In these experiments, uniform value-independent type of noise were introduced by modifying the existing ratings in the data set[2]. More specifically, given a uniform noise profile and density, we have selected a subset of the existing ratings (ignoring "null" values) and altered the existing values – by selecting a completely new rating (which we refer to as *value-independent noise*).

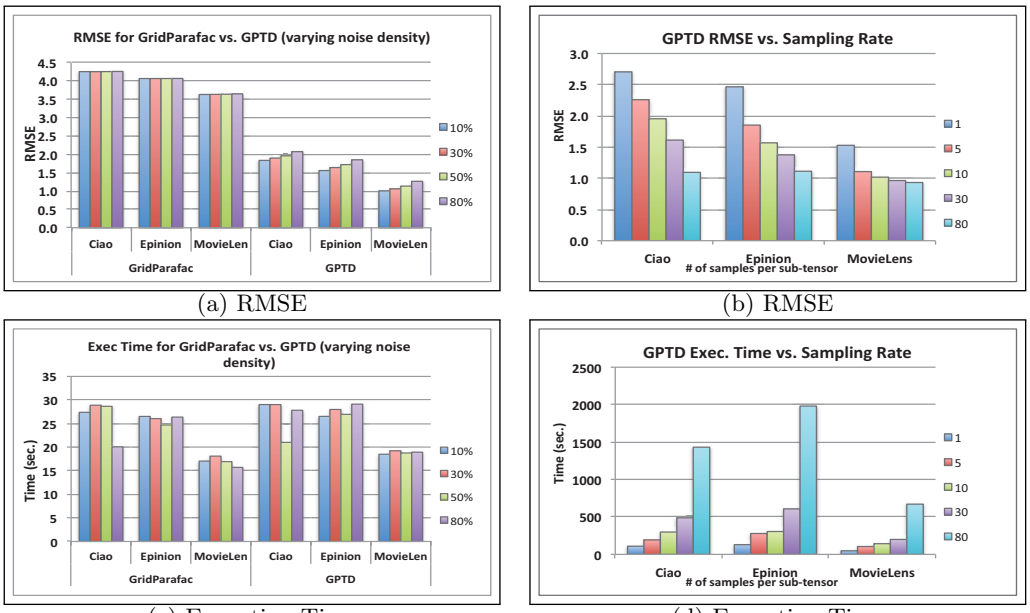
Evaluation Criteria. We use the *root mean squares error* (RMSE) inaccuracy measure to assess the decomposition effectiveness. We also report the decomposition times and memory consumptions. Unless otherwise reported, the execution time of the overall process is reported as if sub-tensor decompositions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

[2]Because of space limitations, we do not include results with slice-concentrated, multi-modal, and value-dependent noise ; but the results for those types of results are similar to the results presented in this section.



(a) RMSE (b) Memory requirement (c) Execution time

Figure 7: GPTD vs. GridParafac alternative (denoted as "Alt"); (uniform noise; value independent noise; noise density 10%; $F = 10$; num. Gibbs samples per sub-tensor = 3; max. num. of P2 iteration = 1000; 4 sub-tensors with noise)



(a) RMSE (b) RMSE

(c) Execution Time (d) Execution Time

Figure 8: **(a)**`GPTD` vs. GridParafac ($2 \times 2 \times 2$ grid; varying noise density; uniform noise; value independent noise; num. Gibbs samples per sub-tensor = 3; $F = 10$; max. num. of P2 iteration = 1000); **(b)** `GPTD` with different num. of Gibbs samples ($4 \times 4 \times 4$ grid; uniform noise; value independent noise; noise density 10%; $F = 10$; max. num. of P2 iteration = 1000

experiment was run 10 times with different random noise distributions across the tensor and averages are reported.

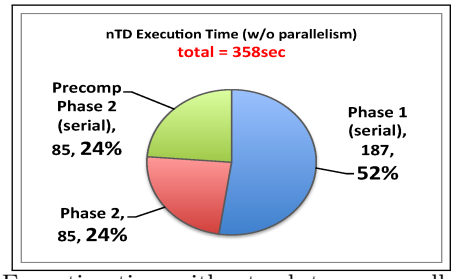
Hardware and Software. We ran experiments on a quad-core CPU Nehalem Node with 12.00GB RAM. All codes were implemented in Matlab and run using Matlab R2015b. For conventional CP decomposition, we used MATLAB Tensor Toolbox Version 2.6 [4].

6.2 Discussion of the Results

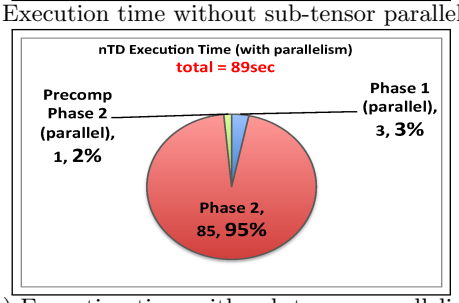
We start the discussion of the results by studying the impact of the `s-strategy` for leveraging noise profiles.

Impact of Leveraging Noise Profiles. In Figure 6, we compare the performance of `nTD` with noise-sensitive sample assignment (i.e., `s-strategy`) against `GPTD` with *uniform* sample assignment and the two naive noise adaptations, presented in Sections 5.1 and 5.2, respectively. Note that in the scenario considered in this figure, we have 640 total Gibbs samples for 64 sub-tensors, providing on the average 10 samples per sub-tensor. In these experiments, we set L_{min} to 9 (i.e. very close to this average), thus requiring that 576(= 64×9) samples are uniformly distributed across the sub-tensors – this leaves only 64 samples to be distributed adaptively across the sub-tensors based on the noise profiles of the sub-tensors and their relationships to other sub-tensors. As we see in this figure, the proposed `nTD` is able to leverage these 64 uncommitted samples to significantly reduced RMSE relative to `GPTD` with *uniform* sample assignment. Moreover, we also see that naive noise adaptations can actually hurt the overall accuracy. `nTD`-naive 1 and 2, both use biased sampling on the noise blocks and focus on the centrality of sub-tensors. Thus, they perform worse than uniform way. These together show that the proposed `s-strategy` is highly effective in leveraging rough knowledge about noise distributions to better allocate the Gibbs samples across the tensor. Note that, as expected, `nTD` is costlier than `GPTD` as it requires additional preprocessing to compute sub-tensor alignments in Phase 2. However, the required pre-processing is trivially parallelizable as discussed next.

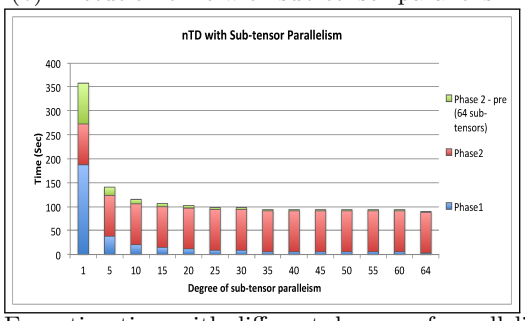
Impact of Sub-Tensor Parallelism. As we see in Figure 9, Phase 1 of the `nTD` algorithm (Algorithm 1) is



(a) Execution time without sub-tensor parallelism



(b) Execution time with sub-tensor parallelism



(c) Execution time with different degrees of parallelism

Figure 9: Impact of sub-tensor parallelism on `nGPTD` ($4\times4\times4$ grid; uniform noise; value independent noise; noise density 10%; $F = 10$; num. Gibbs samples per sub-tensor = 3; max. num. of P2 iteration = 1000; 4 sub-tensors with noise; Ciao data set)

highly parallelizable as the sub-tensors resulting from grid-partitioning can be decomposed in parallel. Similarly, the pre-processing needed for computing the sample assignment strategy in Phase 2 is also highly parallelizable: the most expensive step of the process is the compression of the sub-tensors on modes shared with their neighbors (since the resulting sub-tensor graph is small, the PPR computation has negligible cost) and that work can be done in parallel for each sub-tensor or even for each cell in the resulting compressed representation. Unfortunately, Phase 2, involving incremental stitching and refinement of the factor matrices (see Algorithm 2) cannot be trivially parallelised by assigning different sub-tensors to different processors as the refinement rules need to simultaneously access data from multiple sub-tensors.

GPTD vs. GridParafac in the Presence of Noise In its Phase 1, `nTD` relies on grid based probabilistic decomposition strategy. We next compare this grid probabilistic tensor decomposition (`GPTD`) against the more conventional GridParafac . As we see in Figure 7, `GPTD` provides significantly better accuracy than the conventional approaches and also requires significantly lesser memory. As we expected, we also see that increasing the number of sub-tensors results in a significant drop in the per-sub-tensor memory requirement (therefore improving the scalability of the tensor decomposition process) – though the execution time of the second phase of the process (where the initial decompositions of the sub-tensors are stitched together) increases due to the existence of more sub-tensors to consider.

An important observation in Figure 7 (b) is that the memory requirement for the conventional techniques is very sensitive to data density: While the *MovieLens* tensor has smaller dimensionality then the other two, it has a slightly higher density (3.15×10^{-5} vs. 1.7×10^{-6}). Consequently, for this data set, the memory consumptions of the conventional techniques (especially when the number of grid partitions used are low) are significantly higher than their memory consumptions for the other two data sets. In contrast, the results show that the probabilistic approach is not sensitive to data density and `GPTD` has similar memory usage for all three data sets.

Impact of Noise Density. These results are confirmed in Figure 8(a) & (c), where we vary the noise density between 10% and 80%: as we see here, for all considered noise densities and for all three data sets, the RMSE provided by `GPTD` is significantly better than the RMSE provided by the conventional GridParafac and this RMSE gain does not come with a significant execution time penalty.

Impact of Numbers of Samples. A key parameter of the `GPTD` algorithm is the number of Gibbs samples used per sub-tensor in Phase 1. As we see in Figure 8(b)&(d), as we would expect, increasing the number of Gibbs samples helps reduce the decomposition error (measured using RMSE) ; however having more samples increases the execution time of the algorithm. It is important to note that, when the number of Gibbs samples is low, the algorithm is very fast, indicating that the worst case complexity of the Bayesian iterations arises only when the number of Gibbs samples is very high. Most critically, as we have already seen in Figures 7 and 8(a), the `GPTD` algorithm does not need too many Gibbs samples: using a few (in these experiments, even just 1) Gibbs samples per sub-tensor is sufficient to provide significantly better accuracy than GridParafac, as reported in Figures 7 (a), with similar or better time overhead, as reported in Figure 7 (c).

7. CONCLUSIONS

Web-based user data can be noisy. Recent research has shown that it is possible to improve the resilience of the tensor decomposition process to overfitting (an important challenge in the presence of noisy data) by relying on probabilistic techniques.However, existing techniques assume that all the data and intermediary results can fit in the main memory and (more critically) they treat the entire tensor uniformly, ignoring potential non-uniformities in the noise distribution. In this paper, we proposed a novel *noise-adaptive decomposition* (`nTD`) technique that leverages rough information about noise distribution to improve the tensor decomposition performance. `nTD` partitions the tensor into multiple sub-tensors and then decomposes each sub-tensor probabilistically through Bayesian factorization. The noise profiles of the grid partitions and their alignments are then leveraged to develop a sample assignment strategy (or `s-strategy`) that best suits the noise profile of a given tensor. Experiments with user-centered web data show that `nTD` is significantly better than conventional CP decomposition on noisy tensors.

8. REFERENCES

[1] E. Acar, S. A. Camtepe, M. S. Krishnamoorthy, and B. Yener. Modeling and multiway analysis of chatroom tensors. ISI 2005.

[2] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, N. Tang: Detecting Data Errors: Where are we and what needs to be done? PVLDB 9(12): 993-1004 (2016)

[3] C. A. Andersson and R. Bro. The N-Way Toolbox for Matlab. Chem. and Intel. Lab. Systems, 52(1):1-4, 2000.

[4] B. W. Bader, T. G. Kolda, *et al.* MATLAB Tensor Toolbox Version 2.5, Available online, January 2012.

[5] B. W. Bader, R.A. Harshman, and T.G. Kolda. Temporal analysis of social networks using three-way dedicom. Sandia National Laboratories TR SAND2006-2161, 2006.

[6] R. Balakrishnan, S. Kambhampati: SourceRank: relevance and trust assessment for deep web sources based on inter-source agreement. WWW 2010

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. WWW, 1998.

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

[9] X. Chen and K. S. Candan. LWI-SVD: Low-rank, Windowed, Incremental Singular Value Decompositions on Time-Evolving Data Sets. KDD, 2014.

[10] E. C. Chi and T. G. Kolda Making Tensor Factorizations Robust to Non-Gaussian Noise. tech. report, arXiv: 1010.3043v1, 2010.

[11] W. Chu, Z. Ghahramani, Probabilistic Models for Incomplete Multi-dimensional Arrays. AISTATS 2009.

[12] Xu Chu, Ihab F. Ilyas, P. Papotti, Yin Ye: RuleMiner: Data quality rules discovery. ICDE 2014

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

[14] S. Huang, K. S. Candan, M. L. Sapino. BICP: Block-Incremental CP Decomposition with Update Sensitive Refinement. CIKM 2016

[15] I. Jeon, E. Papalexakis, U. Kang, and C. Faloutsos. HaTen2: Billionscale tensor decompositions. ICDE 2015

[16] B. Jeon, Inah Jeon, Lee Sael and U Kang SCouT: Scalable Coupled Matrix-Tensor Factorization - Algorithm and Discoveries. ICDE 2016.

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

[18] M. Kim and K.S. Candan. Decomposition by normalization (DBN): leveraging approximate functional dependencies for efficient CP and tucker decompositions. Data Min. Knowl. Discov. 30(1): 1-46 (2016)

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

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

[21] M. Leginus, P. Dolog, V. Zemaits. Improving tensor based recommender with clustering. In User Modeling, Adaptation, and Personalization, volume LNCS 7379, pages 151-16. Springer, 2012

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

[23] X. Li, S.Y. Huang, K.S. Candan and M.L. Sapino, 2PCP: Two-phase CP decomposition for billion-scale dense tensors. ICDE 2016.

[24] R. Mehrotra and E. Yilmaz. Terms, topics and tasks: Enhanced user modelling for better personalization. ICTIR 2015

[25] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. PKDD,2012.

[26] A.H. Phan and A. Cichocki, PARAFAC algorithms for large-scale problems, Neurocomputing, vol. 74, no. 11, pp. 1970-1984, 2011.

[27] P. Rai *et al.* Scalable Bayesian Low-Rank Decomposition of Incomplete Multiway Tensors. ICML 2014

[28] R. Rawat. User behaviour modelling in a multidimensional environment for personalization and recommendation. 2010

[29] S. Sadiq, P. Papotti: Big data quality - whose problem is it? ICDE 2016

[30] R. Salakhutdinov and A. Mnih, Probabilistic Matrix Factorization. NIPS 2007

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

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

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

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

[35] 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

[36] L. Xiong, X. Chen, T. K. Huan, J. Schneider J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. SDM 2010.

[37] R. Zafarani, H Liu: Users joining multiple sites: Friendship and popularity variations across sites. Information Fusion 28: 83-89 (2016)

[38] R. Zafarani, L. Tang, Huan Liu: User Identification Across Social Media. TKDD 10(2): 16 (2015)

[39] Q. Zhao, L. Zhang, A. Cichoki, Bayesian CP Factorization of Incomplete Tensors with Automatic Rank Determination. IEEE Trans. on Pat. Analysis and Mach. Intel. 2014.

[40] http://grouplens.org/datasets/movielens/

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