TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Efficient Optimization Algorithms for Large Tensor Processing

by

Christos Kolomvakis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE MASTER OF SCIENCE OF

ELECTRICAL AND COMPUTER ENGINEERING

November 2021

THESIS COMMITTEE

Professor Athanasios P. Liavas, *Thesis Supervisor*
Professor George N. Karystinos
Associate Professor Vasilis Samoladas

Abstract

In this thesis, we consider the problem of tensor completion. We investigate two cases: In the first part, we consider Nonnegative Tensor Completion. We propose an improvement over an existing distributed algorithm for the solution of this problem, test it on synthetic and real datasets, and measure the execution time and speedups.

In the second part, we consider unconstrained tensor completion with smoothing constraints. We present the problem statement and we propose a distributed algorithm for its solution. We develop an algorithm which takes into account the distribution of the nonzero elements during the assignment of subtensors (and, as a result, of the corresponding subfactors) to each processor. We test our adaptive partitioning algorithm on real world datasets and measure the attained speedup.

Acknowledgements

As I am nearing the end of my M.Sc. studies, I would like to thank people that helped me with this thesis.

First, I would like to thank my supervisor, Professor Athanasios Liavas, for his guidance throughout this work.

The M.Sc. experience would not be as special as it was, without my labmates: Ioannis - Marios Papagiannakos, Ioanna Siaminou, Paris Karakasis and Margarita Psychountaki.

I would like to thank my parents and my brothers for their support.

Special thanks goes to Ms. Elpida for her support.

Finally, I want to thank my friends for their support and all the nice moments we shared.

Table of Contents

List of Abbreviations

ADMM	Alternating Direction Method of Multipliers
ALS	Alternating Least Squares
AO	Alternating Optimization
AOO	All-at-Once Optimization
BSUM	Block Successive Upper bound Minimization
CANDECOMP	Canonical Decomposition
CPD	Canonical Polyadic Decomposition
CSID	Canonical System Identification
KKT	Karush-Kuhn-Tucker
i.i.d.	independent and identically distributed
LS	Least Squares
MNLS	Matrix Nonnegative Least Squares
MLSME	Matrix Least Squares with Missing Elements
MPI	Message Passing Interface
MTTKRP	Matricized Tensor Khatri - Rao Product
MU	Multiplicative Update
NMF	Nonnegative Matrix Factorization
NMLSME	Nonnegative Matrix Least Squares with Missing Elements
NNLS	Nonnegative Least Squares

NTC	Nonnegative Tensor Completion
NTF	Nonnegative Tensor Factorization
PARAFAC	Parallel Factor Analysis
RFE	Relative Factor Error
SVD	Singular Value Decomposition

List of Figures

List of Tables

Chapter 1

Introduction

Tensors are mathematical objects that have recently gained great popularity due to their ability to model multiway data dependencies [2], [3], [1], [4]. Tensor factorization (or decomposition) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization and interpretation. Tensor factorizations are usually computed as solutions of optimization problems [2], [3]. Alternating Optimization (AO), All–at–Once Optimization (AOO), and Multiplicative Updates (MUs) are among the most frequently used techniques to solve those optimization problems. Some of the most widely used tensor factorization models include the Canonical Polyadic Decomposition (CPD or CANDECOMP), the Tucker Decomposition and the Tensor Train Decomposition [5]. CPD may also be referred to as Parallel Factor Analysis (PARAFAC). The problem of calculating a tensor factorization with nonnegativity constraints is known as Nonnegative Tensor Factorization (NTF). In this work, we focus on nonnegative tensor completion, using the PARAFAC model, and unconstrained tensor completion, where we also use the PARAFAC model with smoothing constraints.

Recent Work for constrained tensor factorization/completion includes, among others, [6], [7], [8], [9] and [10]. Stochastic methods have also been used for the computation of Tensor Factorizations. Recent work on Stochastic Factorization/Completion includes [11], [12], [13], [14] and [15].

In [6], several NTF algorithms and a detailed convergence analysis have been developed. A general framework for joint matrix/tensor factorization/completion has been developed in [7]. In [8], an Alternating Direction Method of Multipliers (ADMM) algorithm for NTF has been derived, and an architecture for its parallel implementation has been outlined. In [9], the authors consider constrained matrix/tensor factorization/completion problems. They adopt the AO framework as outer loop and use the ADMM for solving the inner constrained optimization problem for one matrix factor conditioned on the rest.

In [16], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations on a multi–core system have been reported. Related work on parallel algorithms for sparse tensor decomposition includes [17] and [18].

Prior works to the smooth tensor factorization problem, include [19], [20] and [21], while works that use smooth tensor decompositions for tensor completion include [22] and [23]. The authors in [24] utilize tensor completion with smoothing constraints to approximate a function from data points. To the best of our knowledge, there is no distributed algorithm for the solution of smooth CPD.

1.1 Our contribution

In this work, we focus on tensor completion problems. We study Nonnegative tensor completion and unconstrained tensor completion problems with smoothing constraints. Our aim is to derive efficient algorithms, suitable for parallel implementation. We adopt the AO framework and solve each Matrix Least Squares (MLS) problem; depending on the constraints, we either solve an unconstrained Matrix Least Squares problem, or a Nonnegative Matrix Least Squares problem via a first order optimal (Nesterov–type) algorithm for L–smooth μ–strongly convex problems.[1] Then, we describe in detail MPI implementations of the AO algorithms and measure the speedup attained in a multi–core environment.

1.2 Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital bold letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. Their elements are denoted by small nonbold letters and a set of indices, for example, $x_{i,j}$. For a matrix or a tensor, we may also denote an element by $[\mathbf{X}]_{i,j}$ and $[\boldsymbol{\mathcal{X}}]_{i,j,k}$, respectively, for convenience. We denote the i–th column of a matrix $\mathbf{X}$ as $\mathbf{x}_i$.

Sets are denoted by blackboard bold capital letters; for example, $\mathbb{U}$. $\mathbb{R}$ denotes the set of real numbers. $\mathbb{R}_+^{I\times J\times K}$ denotes the set of $(I \times J \times K)$ real nonnegative tensors, while $\mathbb{R}_+^{I\times J}$ denotes the set of $(I \times J)$ real nonnegative matrices. $\|\cdot\|_F$ denotes the Frobenius norm of the tensor or matrix argument, $\mathbf{I}$ denotes the identity matrix of appropriate dimensions, and $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of element–wise nonnegative matrices.

The outer product of vectors $\mathbf{a} \in \mathbb{R}^{I\times 1}$, $\mathbf{b} \in \mathbb{R}^{J\times 1}$, and $\mathbf{c} \in \mathbb{R}^{K\times 1}$ is the rank-one tensor $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I\times J\times K}$ with elements $[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}]_{i,j,k} = \mathbf{a}_i\mathbf{b}_j\mathbf{c}_k$. The Khatri-Rao (columnwise Kronecker) product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \odot \mathbf{B}$, the Kronecker

[1] We note that a closely related algorithm for the solution of MNLS problems has been used in [25] and [26]; we explain in detail later the performance improvement offered by our approach.

product is denoted as $\mathbf{A} \otimes \mathbf{B}$ and the Hadamard (elementwise) product is denoted as $\mathbf{A} \circledast \mathbf{B}$.

Finally, inequality $\mathbf{A} \succeq \mathbf{B}$ means that matrix $\mathbf{A} - \mathbf{B}$ is positive semidefinite, and by $\mathbf{X} \geq \mathbf{0}$, we denote a matrix $\mathbf{X}$ which has nonnegative elements.

1.3 Thesis Outline

The thesis is organized as follows:

- In Chapter 2, we introduce some background on Matrix Least Squares problems. We discuss the unconstrained problem, and devote most of the section to the Nonnegative Matrix Least Squares problem, where we analyze a first order accelerated gradient method to get an approximate solution.
- In Chapter 3, we discuss the CP Decomposition. We start by giving some mathematical background on tensors, and proceed to present the unconstrained and nonnegative PARAFAC models. We make a brief mention to the algorithms that compute the models.
- Chapter 4 is devoted to Nonnegative Tensor Completion (NTC). We present the problem statement, and explain Nonnegative Matrix Least Squares with Missing Elements, which is the building block for the solution of the NTC problem. We present a new parallel scheme for the solution of NTC and show experiments that measure the speedup of our algorithm.
- Chapter 5 is dedicated to unconstrained tensor completion with smoothing constraints. We discuss the problem and its solution, and we present a parallel scheme to solve this problem. We develop an adaptive partitioning algorithm that can lead to improvement of speedup in datasets which do not have their nonzeros distributed uniformly.
- Finally, in Chapter 6, we conclude the thesis and suggest future work.

Chapter 2

Matrix Least Squares Problems

In this chapter, we describe the matrix least squares problem, which will be the workhorse towards the development of efficient algorithms for tensor factorizations. We first give some definitions related to linear algebra operations.

2.1 Mathematical Background

Definition 2.1 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{P\times K}$*. The* ***Kronecker product*** *(or tensor product) of* $\mathbf{A}$ *and* $\mathbf{B}$ *is defined as the matrix*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,M}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{N,1}\mathbf{B} & \cdots & a_{N,M}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{NP\times MK}. \tag{2.1}$$

Definition 2.2 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{P\times M}$*. The* ***Khatri–Rao product*** *of* $\mathbf{A}$ *and* $\mathbf{B}$ *is defined as the matrix*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \cdots & \mathbf{a}_M \otimes \mathbf{b}_M \end{bmatrix} \in \mathbb{R}^{NP\times M}. \tag{2.2}$$

Definition 2.3 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{N\times M}$*. The* ***Hadamard Product*** *or elementwise matrix product of* $\mathbf{A}$ *and* $\mathbf{B}$ *is a matrix of size* $N \times M$*, and is defined as*

$$[\mathbf{A} \circledast \mathbf{B}]_{n,m} = a_{n,m}b_{n,m}, \tag{2.3}$$

for all $n \in \{1, \ldots, N\}$, $m \in \{1, \ldots, M\}$.

Regarding the Kronecker and Khatri–Rao products, we note that they are both associative [4]. For example, a Khatri–Rao product of three matrices can be equivalently calculated as

$$\mathbf{A} \odot \mathbf{B} \odot \mathbf{C} = (\mathbf{A} \odot \mathbf{B}) \odot \mathbf{C} = \mathbf{A} \odot (\mathbf{B} \odot \mathbf{C}). \tag{2.4}$$

2.2 Matrix Least Squares

Let $\mathbf{X} \in \mathbb{R}^{M\times N}$ and $\mathbf{B} \in \mathbb{R}^{N\times R}$ and consider the problem

$$\min_{\mathbf{A}} f(\mathbf{A}) = \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2. \tag{2.5}$$

This problem is known as the matrix least squares problem. The cost function is convex, therefore, the existence of a global minimizer of f, $\mathbf{A}^*$, is guaranteed. The solution $\mathbf{A}^*$ must satisfy the following system of linear equations

$$\mathbf{X}\mathbf{B} = \mathbf{A}^*\mathbf{B}^T\mathbf{B}, \tag{2.6}$$

which are known as *normal equations*. Thus, $\mathbf{A}^*$ is given by

$$\mathbf{A}^* = \mathbf{X}\mathbf{B}^\dagger = \mathbf{X}\mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}. \tag{2.7}$$

For an extensive discussion on the computational aspects of the solution of the normal equations the reader is referred to [27].

2.3 Matrix Nonnegative LS

Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{A} \in \mathbb{R}_+^{m\times r}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$, and consider the Matrix Nonnegative LS (MNLS) problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2. \tag{2.8}$$

The problem given in (2.8), is convex, however, it does not have a closed form solution. Hence, we resort to iterative methods that solve efficiently problems of this form in the following sections. The next section focuses on L–smooth μ–strongly convex optimization problems.

2.3.1 Optimal first–order methods for L–smooth μ–strongly convex optimization problems

We consider optimization problems of smooth and strongly convex functions and briefly present results concerning their information complexity and the associated first–order optimal algorithms (for a detailed exposition see [28, Chapter 2]).

We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth (that is, differentiable up to a sufficiently high

Algorithm 1: Accelerated gradient algorithm for L-smooth μ-strongly convex problems.

Input: $\mathbf{x}_0 \in \mathbb{R}^N$, μ, L. Set $\mathbf{y}_0 = \mathbf{x}_0$, $\mathcal{K} = \frac{L}{\mu}$, $\beta = \frac{\sqrt{\mathcal{K}}-1}{\sqrt{\mathcal{K}}+1}$.

1 k-th iteration

2 $\quad \mathbf{x}_{k+1} = \left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right)_{\mathbb{X}}$

3 $\quad \mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta(\mathbf{x}_{k+1} - \mathbf{x}_k)$

order) convex function, with gradient $\nabla f(\mathbf{x})$ and Hessian $\nabla^2 f(\mathbf{x})$. Our aim is to solve the problem

$$\min_{\mathbf{x}} f(\mathbf{x}), \tag{2.9}$$

within accuracy $\epsilon > 0$. The solution accuracy is defined as follows. If $f^* := \min_{\mathbf{x}} f(\mathbf{x})$, then point $\bar{\mathbf{x}} \in \mathbb{R}^n$ solves problem (2.9) within accuracy ϵ if $f(\bar{\mathbf{x}}) - f^* \leq \epsilon$.

Let $0 < \mu \leq L < \infty$. A smooth convex function f is called L–smooth or, using the notation of [28, p. 66], $f \in \mathcal{S}^{\infty,1}_{0,L}$, if

$$\mathbf{0} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall \mathbf{x} \in \mathbb{R}^n, \tag{2.10}$$

and L–smooth μ–strongly convex, or $f \in \mathcal{S}^{\infty,1}_{\mu,L}$, if

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{2.11}$$

The number of iterations that first-order methods need for the solution of problem (2.9), within accuracy ϵ, is $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ if $f \in \mathcal{S}^{\infty,1}_{0,L}$, and $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ if $f \in \mathcal{S}^{\infty,1}_{\mu,L}$ [28, Theorem 2.2.2]. The convergence rate in the first case is *sublinear* while, in the second case, it is *linear* and determined by the condition number of the problem, $\mathcal{K} := \frac{L}{\mu}$. Thus, strong convexity is a very important property that should be exploited whenever possible.

An algorithm that achieves this complexity, and, thus, is first–order optimal, appears in Algorithm 1 (see, also [28, p. 80]).

If the problem of interest is the constrained problem

$$\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x}), \tag{2.12}$$

where $\mathbb{X}$ is a closed convex set, then the corresponding optimal algorithm is very much alike Algorithm 1, with the only difference being in the computation of $\mathbf{x}_{k+1}$. We now have

that [28, p. 90]

$$\mathbf{x}_{k+1} = \Pi_{\mathbb{X}} \left(\mathbf{y}_k - \frac{1}{L} \nabla f(\mathbf{y}_k) \right), \tag{2.13}$$

where $\Pi_{\mathbb{X}}(\cdot)$ denotes the Euclidean projection onto set $\mathbb{X}$. The convergence properties of this algorithm are the same as those of Algorithm 1. If the projection onto set $\mathbb{X}$ is easy to compute, then the algorithm is both theoretically optimal and very efficient in practice.

2.3.2 Optimal first–order methods for L–smooth μ–strongly convex MNLS problems

Algorithm 2: Accelerated gradient algorithm for NMLS problems with proximal term.

Input: $\mathbf{X} \in \mathbb{R}^{P \times Q}$, $\mathbf{B} \in \mathbb{R}^{Q \times R}$, $\mathbf{A}_* \in \mathbb{R}^{P \times R}$
$L = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu = \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$
$\lambda = g(L, \mu)$, $\mathcal{K} = \frac{L+\lambda}{\mu+\lambda}$, $\beta = \frac{\sqrt{\mathcal{K}}-1}{\sqrt{\mathcal{K}}+1}$
$\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
$\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
$k = 0$
while (terminating condition is FALSE) **do**
  $\nabla f_{\mathrm{P}}(\mathbf{Y}_k) = \mathbf{W} + \mathbf{Y}_k\mathbf{Z}$
  $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda} \nabla f(\mathbf{Y}_k)\right)_+$
  $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta\,(\mathbf{A}_{k+1} - \mathbf{A}_k)$
  $k = k + 1$
return $\mathbf{A}_k$.

In this section, we present an optimal first–order algorithm for the solution of L–smooth μ–strongly convex MNLS problems. Optimal first–order methods have recently attracted great research interest because they are strong candidates and, in many cases, the only viable way for the solution of very large optimization problems.

Nesterov–type algorithm for MNLS with proximal term

In the sequel, we present a Nesterov–type algorithm for the MNLS problem with proximal term. Let $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{A} \in \mathbb{R}^{m \times r}$, $\mathbf{B} \in \mathbb{R}^{n \times r}$, $\lambda > 0$ and consider the problem

$$\min_{\mathbf{A} \geq \mathbf{0}} f(\mathbf{A}) := \frac{1}{2} \|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2 + \frac{\lambda}{2} \|\mathbf{A} - \mathbf{A}_*\|_F^2. \tag{2.14}$$

The gradient and Hessian of f, at point $\mathbf{A}$, are, respectively,

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A} - \mathbf{A}_*) \tag{2.15}$$

and

$$\nabla^2 f(\mathbf{A}) := \frac{\partial^2 f(\mathbf{A})}{\partial \text{vec}(\mathbf{A}) \partial \text{vec}(\mathbf{A})^T} = \mathbf{B}^T\mathbf{B} \otimes \mathbf{I} + \lambda \mathbf{I} \succeq \mathbf{0}. \tag{2.16}$$

Let $L := \max(\text{eig}(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}))$ and $\mu := \min(\text{eig}(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}))$. Since $\mu > 0$, the problem (2.14) is L–smooth μ-strongly convex. A first–order optimal algorithm for the solution of (2.14) can be derived using the approach of Section 2.3.1. We note that [25] and [26] solved problem (2.14) using a variation of Algorithm 1, which is equivalent to Algorithm 1 with $\mu = 0$. However, if $\mu > 0$, then this algorithm is *not* first–order optimal and, as we shall see later, it performs much worse than the optimal.

We note that the values of L and μ are necessary for the development of the Nesterov-type algorithm, thus, their computation is imperative.[1]

We choose λ based on L and μ, and denote this functional dependence as $\lambda = g(L, \mu)$. If $\frac{\mu}{L} \ll 1$, then we may set $\lambda \approx 10\mu$, significantly improving the conditioning of the problem by putting large weight on the proximal term; however, in this case, we expect that the optimal point will be biased towards $\mathbf{A}_*$. Otherwise, we may set $\lambda \lessapprox \mu$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible.

The Karush–Kuhn–Tucker (KKT) conditions for problem (2.14) are [25]

$$\nabla f(\mathbf{A}) \geq \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \nabla f(\mathbf{A}) \circledast \mathbf{A} = \mathbf{0}. \tag{2.17}$$

These expressions can be used in a terminating condition. For example, we may terminate the algorithm if

$$\min_{i,j}\left([\nabla f(\mathbf{A})]_{i,j}\right) > -\delta_1, \max_{i,j}\left(\left|[\nabla f(\mathbf{A}) \circledast \mathbf{A}]_{i,j}\right|\right) < \delta_2, \tag{2.18}$$

for small positive real numbers δ_1 and δ_2. Of course, other criteria, based, for example, on the (relative) change of the cost function can be used in terminating conditions.

A Nesterov–type algorithm for the solution of the MNLS problem with proximal term

[1] An alternative to their direct computation is to estimate L using line-search techniques and overcome the computation of μ using heuristic adaptive restart techniques [29]. However, in our case, this alternative is computationally demanding, especially for large–scale problems, and shall not be considered.

(2.14) is given in Algorithm 2. For notational convenience, we denote Algorithm 2 as

$$\mathbf{A}_{\text{opt}} = \text{Nesterov\_MNLS}(\mathbf{X}, \mathbf{B}, \mathbf{A}_*).$$

Computational complexity of Algorithm 2

Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mnr)$ and $O(rn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(r^3)$ operations. $\nabla f(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration with cost $O(mr^2)$, $O(mr)$, and $O(mr)$ arithmetic operations, respectively.

Chapter 3

Tensor Factorizations

In this chapter, we will discuss the CP Decomposition. First, we will give some preliminary definitions about tensors.

3.1 Definitions

Definition 3.1 *A tensor is a multidimensional array. The* ***order*** *of a tensor defines the number of dimensions a tensor has.*

Based on the definition above, a third order tensor is a tensor with three indices. An illustration is shown in Figure 3.1.

A matrix is a second–order tensor and a vector is a first–order tensor. We collectively refer to tensors of order three or higher as higher–order tensors.

Definition 3.2 *The* ***Frobenius norm*** *of a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *is defined as*

$$\|\boldsymbol{\mathcal{X}}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1,i_2,\ldots,i_N}^2}. \tag{3.1}$$

Definition 3.3 *A* ***fiber*** *of a tensor is a vector defined by fixing all indices of a tensor but one.* As an example, let us assume a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. $x_{:,j,k}$ defines a mode–1 fiber,

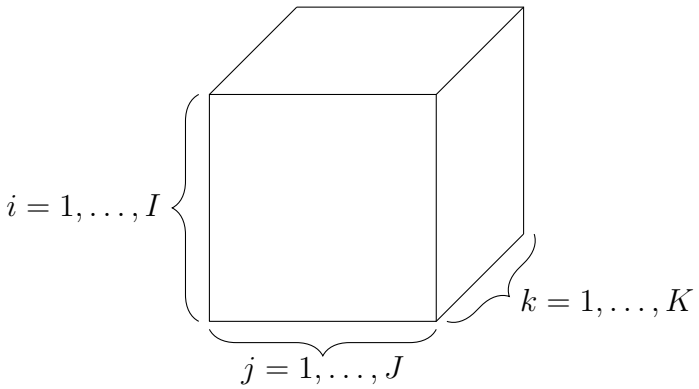

Figure 3.1: A third order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$.

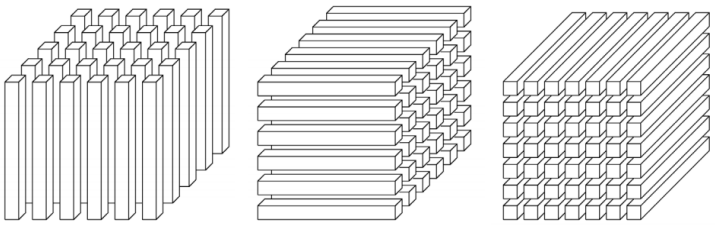

Figure 3.2: Mode–1, mode–2 and mode–3 fibers for a third order tensor, respectively. This image is derived from [1].

$x_{i,:,k}$ defines a mode–2 fiber and $x_{i,j,:}$ defines a mode–3 fiber, as shown in the figure below.

Definition 3.4 *Let* $\mathbf{a} \in \mathbb{R}^N$, $\mathbf{b} \in \mathbb{R}^P$, *and* $\mathbf{c} \in \mathbb{R}^J$. *The* ***outer product*** *of* $\mathbf{a}$ *and* $\mathbf{b}$ *is defined as the* ***rank–one matrix*** *with elements*

$$[\mathbf{a} \circ \mathbf{b}]_{n,p} = a_n b_p, \tag{3.2}$$

for all $n \in \{1, \ldots, N\}$, $p \in \{1, \ldots, P\}$. *In the same manner, the outer product of* $\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$ *is defined as the* ***rank–one tensor*** *with elements*

$$[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}]_{n,p,j} = a_n b_p c_j, \tag{3.3}$$

for all $n \in \{1, \ldots, N\}$, $p \in \{1, \ldots, P\}$, *and* $j \in \{1, \ldots, J\}$. Rank–1 tensors for N–th order tensors, with $N > 3$, are defined similarly.

Definition 3.5 *Let* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. *The* ***matricization of the tensor with respect to the*** j***–th mode*** *(mode–*j *matricization) is defined as the matrix* $\mathbf{X}_{(j)} \in \mathbb{R}^{I_j \times \prod_{k=1, k \neq j}^{N} I_k}$, *where the element* $[\boldsymbol{\mathcal{X}}]_{i_1, i_2, \ldots, i_j, \ldots, i_N}$ *is mapped to* $[\mathbf{X}_{(j)}]_{i_j, k}$ *according to [1]*:

$$k = 1 + \sum_{\substack{p=1 \\ p \neq j}}^{N} (i_p - 1) J_p, \text{ where } J_p = \prod_{\substack{m=1 \\ m \neq j}}^{p-1} I_m.$$

This can be better understood through an example. Let a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. We are interested in examining the mode–1 matricization of $\boldsymbol{\mathcal{X}}$, $\mathbf{X}_{(1)} \in \mathbb{R}^{I \times JK}$. Then, element $[\boldsymbol{\mathcal{X}}]_{i_1, i_2, i_3}$ will be mapped to element $[\mathbf{X}_{(1)}]_{i_1, k}$, where

$$k = 1 + (i_2 - 1) + (i_3 - 1) I_2, \tag{3.4}$$

since $\prod_{m=2}^{1} I_m = \prod_{m \in \emptyset} I_m = 1$ (by convention) and $J_3 = \prod_{m=2}^{2} I_m = I_2$. For the mode–2

matricization, $\mathbf{X}_{(2)} \in \mathbb{R}^{J \times IK}$, element $[\boldsymbol{\mathcal{X}}]_{i_1,i_2,i_3}$ will be mapped to element $[\mathbf{X}_{(2)}]_{i_2,k'}$, where

$$k' = 1 + (i_1 - 1) + (i_3 - 1)I_1,$$

since $J_1 = \prod_{m=1}^{0} I_m = 1$, $J_3 = \prod_{m=2}^{2} I_m = I_1$, and so on.

Definition 3.6 *The Canonical Polyadic (CP) or PARAFAC decomposition of a tensor is given by*

$$\boldsymbol{\mathcal{X}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] := \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \cdots \circ \mathbf{u}_r^{(N)}. \tag{3.5}$$

It has been shown that under mild conditions, a tensor rank decomposition is unique, up to a scaling and permutation ambiguity.

3.2 PARAFAC model

Let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{U}^{o(1)}, \ldots, \mathbf{U}^{o(N)}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{o(1)} \circ \cdots \circ \mathbf{u}_r^{o(N)}, \tag{3.6}$$

where $\mathbf{U}^{o(i)} = [\mathbf{u}_1^{o(i)} \ \cdots \ \mathbf{u}_R^{o(i)}] \in \mathbb{R}^{I_i \times R}$, with $i \in \{1, \ldots, N\}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Estimates of $\mathbf{U}^{o(i)}$ can be obtained by computing matrices $\mathbf{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, for $i \in \{1, \ldots, N\}$, that solve the optimization problem

$$\min_{\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}} f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right), \tag{3.7}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization. A common choice for $f_{\boldsymbol{\mathcal{X}}}$ is

$$f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right\|_F^2. \tag{3.8}$$

If $\boldsymbol{\mathcal{Y}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$, then, for an arbitrary mode i, the corresponding matrix unfolding is given by [1]

$$\begin{aligned} \mathbf{Y}_{(i)} &= \mathbf{U}^{(i)} \left(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(i+1)} \odot \mathbf{U}^{(i-1)} \odot \cdots \odot \mathbf{U}^{(1)}\right)^T, \\ \mathbf{Y}_{(i)} &= \mathbf{U}^{(i)} \mathbf{K}^{(i)^T}, \end{aligned} \tag{3.9}$$

where we define $\mathbf{K}^{(i)}$ as

$$\mathbf{K}^{(i)} = \left(\bigodot_{j=i+1}^{N} \mathbf{U}^{(j)} \odot \bigodot_{j=1}^{i-1} \mathbf{U}^{(j)} \right). \tag{3.10}$$

Thus, $f_{\boldsymbol{\mathcal{X}}}$ can be expressed as

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) = \frac{1}{2} \left\| \mathbf{X}_{(i)} - \mathbf{Y}_{(i)} \right\|_F^2 \tag{3.11}$$

These expressions form the basis of the alternating least squares algorithm (ALS) for tensor factorization, in the sense that, for fixed matrix factors $\mathbf{U}^{(j)}$, with $j \neq i$, we can update $\mathbf{U}^{(i)}$ by solving a matrix least–squares problem.

3.3 Constrained Tensor Decomposition

In many applications, we are interested in tensor decompositions whose factors should comply with constraints emerging from underlying models or for interpretability reasons. Specifically, let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{U}^{(1)o}, \ldots, \mathbf{U}^{(N)o}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{(1)o} \circ \cdots \circ \mathbf{u}_r^{(N)o}, \tag{3.12}$$

where $\mathbf{U}^{(n)o} = \begin{bmatrix} \mathbf{u}_1^{(n)o} & \cdots & \mathbf{u}_R^{(n)o} \end{bmatrix} \in \mathbb{B}^n \subseteq \mathbb{R}^{I_n \times R}$, with $n \in \{1, \ldots, N\}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the additive noise. Then, the problem of finding estimates of the factors $\mathbf{U}^{(n)o}$ can be formulated as

$$\begin{aligned} \min_{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n)}} & f_{\boldsymbol{\mathcal{X}}} \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right) \\ \text{s.t.} \quad & \mathbf{U}^{(n)} \in \mathbb{B}^n, \quad n \in \{1, \ldots, N\}, \end{aligned} \tag{3.13}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization. As in the unconstrained case, we focus on the sum of squared errors cost function

$$f_{\boldsymbol{\mathcal{X}}} \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right\|_F^2. \tag{3.14}$$

Under the ALS framework, each factor can be updated via solving an unconstrained/constrained matrix least–squares problem. It can be formulated as:

$$\begin{aligned} \min_{\mathbf{U}^{(i)}} \quad & \frac{1}{2}\|\mathbf{X}_{(i)} - \mathbf{U}^{(i)}\mathbf{K}^{(i)}\|_F^2 \\ \text{s.t.} \quad & \mathbf{U}^{(i)} \in \mathbb{B}^i, \quad i \in \{1, \ldots, N\}. \end{aligned} \tag{3.15}$$

Set $\mathbb{B}^i$, for $i \in \{1, \ldots, N\}$, can be

- $\mathbb{R}^{I_i \times R}$: unconstrained case,
- $\mathbb{R}_+^{I_i \times R}$: case of nonnegativity constraints.

Tensor Decomposition with nonnegativity constraints

Recall expressions (3.7), (3.8) and (3.11):

$$\begin{gathered} \min_{\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}} f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right), \\ f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2}\left\|\boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]\right\|_F^2, \\ f_{\boldsymbol{\mathcal{X}}}(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) = \frac{1}{2}\left\|\mathbf{X}_{(i)} - \mathbf{Y}_{(i)}\right\|_F^2. \end{gathered}$$

If we introduce the constraint $\{\mathbf{U}^{(i)}\}_{i=1}^N \geq \mathbf{0}$, then these expressions form the basis for the AO NTF in the sense that, if we fix all but one of the matrix factors, we can update the remaining factor by solving an MNLS problem.

It has been shown in [30] that the AO NTF algorithm with proximal term falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (3.7).

Chapter 4

Tensor Completion

4.1 Introduction

In many cases of practical interest, we observe a small subset of the elements of tensor $\mathcal{X}$, indexed by $\Omega \subseteq \mathbb{N}_{I_1} \times \cdots \times \mathbb{N}_{I_N}$. Let $\mathcal{M}$ be a binary tensor with the same size as $\mathcal{X}$ whose elements are defined as

$$\mathcal{M}(i_1, i_2, \ldots, i_N) = \begin{cases} 1, & \text{if } (i_1, i_2, \ldots, i_N) \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The number of nonzero elements of $\mathcal{X}$ is equal to $\texttt{nnz} := |\Omega|$. The TC problem can be expressed as

$$\min f_\Omega\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) + \frac{\lambda}{2} \sum_{i=1}^{N} \|\mathbf{U}^{(i)}\|_F^2, \tag{4.2}$$

where

$$f_\Omega\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\| \mathcal{M} \circledast \left(\mathcal{X} - [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\right) \right\|_F^2. \tag{4.3}$$

If $\mathcal{Y} = [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$, then

$$f_\Omega\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\| \mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{Y}_{(i)}\right) \right\|_F^2, \quad i \in \mathbb{N}_N, \tag{4.4}$$

where $\mathbf{M}_{(i)}$, $\mathbf{X}_{(i)}$, and $\mathbf{Y}_{(i)}$ are, respectively, the matrix unfoldings of $\mathcal{M}$, $\mathcal{X}$, and $\mathcal{Y}$ with respect to the i-th mode. Similarly to the dense case, these expressions form the basis of the AO method for TC. More specifically, if we consider factor $\mathbf{U}^{(i)}$ as a variable, with all the other factors being fixed, then we can update $\mathbf{U}^{(i)}$ by solving the problem

$$\min_{\mathbf{U}^{(i)} \in \mathbb{U}^{(i)}} \left\| \mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{U}^{(i)} \mathbf{K}^{(i)T}\right) \right\|_F^2 + \frac{\lambda}{2} \|\mathbf{U}^{(i)}\|_F^2. \tag{4.5}$$

4.2 Nonnegative Matrix Least Squares with Missing Elements

The solution of the Matrix Least Squares with Missing Elements (MLSME) problem will be the building block for the solution of the TC problem. Let $\mathbf{X} \in \mathbb{R}^{P\times Q}$, $\mathbf{A} \in \mathbb{R}^{P\times R}$, and $\mathbf{B} \in \mathbb{R}^{Q\times R}$. Let $\Omega \subseteq \mathbb{N}_P \times \mathbb{N}_Q$ be the set of indices of the known elements of $\mathbf{X}$ and let $\mathbf{M}$ be a matrix with the same size as $\mathbf{X}$, with elements $\mathbf{M}(i,j)$ equal to one or zero, based on the availability of the corresponding element of $\mathbf{X}$. We consider the problem

$$\min_{\mathbf{A}\in\mathbb{A}} f_\Omega(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{M} \circledast \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\right\|_F^2 + \frac{\lambda}{2}\|\mathbf{A}\|_F^2. \tag{4.6}$$

The gradient and the Hessian of f_Ω, at point $\mathbf{A}$, are given by

$$\nabla f_\Omega(\mathbf{A}) = -\left(\mathbf{M} \circledast \mathbf{X} - \mathbf{M} \circledast \left(\mathbf{A}\mathbf{B}^T\right)\right)\mathbf{B} + \lambda\mathbf{A} \tag{4.7}$$

and

$$\nabla^2 f_\Omega(\mathbf{A}) = \left(\mathbf{B}^T \otimes \mathbf{I}_P\right) \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{M}\right)\right)\left(\mathbf{B} \otimes \mathbf{I}_P\right) + \lambda\mathbf{I}_{PR}. \tag{4.8}$$

We focus on the nonnegative case, where we solve the MLSME problem using the Nesterov-type algorithm of Algorithm 3. We observe that this algorithm is much more complicated than Algorithm 2, mainly because of the computations in line 6, which must be repeated in every iteration.

A crucial point of the algorithm is the assignment of values to parameters μ and L. If we denote the optimal values as μ^* and L^*, then it turns out that $\mu^* + \lambda$ and $L^* + \lambda$ are, respectively, equal to the smallest and the largest eigenvalue of $\nabla^2 f_\Omega$. As the size of the problem grows, the computation of μ^* and L^* becomes very demanding. We set $\mu = 0$ and $L = \max(\operatorname{eig}(\mathbf{B}^T\mathbf{B}))$, which can be easily computed, especially in the cases of small R. We have observed that, in practice, our choice for μ is very accurate for very sparse problems, while our choice for L is an easily computed upper bound for L^*.

We denote the output of Algorithm 3 as

$$\mathbf{A}^+ = \mathrm{NMLSME}(\mathbf{X}, \mathbf{M}, \mathbf{B}, \mathbf{A}_*, \lambda).$$

The computational complexity of Algorithm 3 is as follows:

1. the computation of $\mathbf{W}$ requires $O(|\Omega|R)$ arithmetic operations;
2. the computation of $\mathbf{Z}_l$ requires $O\left(|\Omega|R\right)$ arithmetic operations;

Algorithm 3: Nesterov-type algorithm for the nonnegative MLSME problem.

Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}^{P\times Q}$, $\mathbf{B} \in \mathbb{R}^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}^{P\times R}$, λ, μ, L

1 $\mathbf{W} = -(\mathbf{M} \circledast \mathbf{X})\mathbf{B}$
2 $\mathcal{K} = \frac{L+\lambda}{\mu+\lambda}$, $\beta = \frac{\sqrt{\mathcal{K}}-1}{\sqrt{\mathcal{K}}+1}$
3 $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
4 $l = 0$
5 **while** (*terminating condition is FALSE*) **do**
6 $\quad \mathbf{Z}_l = \left(\mathbf{M} \circledast \left(\mathbf{Y}_l\mathbf{B}^T\right)\right)\mathbf{B}$
7 $\quad \nabla f_\Omega(\mathbf{Y}_l) = \mathbf{W} + \mathbf{Z}_l + \lambda\mathbf{Y}_l$
8 $\quad \mathbf{A}_{l+1} = \left(\mathbf{Y}_l - \frac{1}{L+\lambda}\nabla f_\Omega(\mathbf{Y}_l)\right)_+$
9 $\quad \mathbf{Y}_{l+1} = \mathbf{A}_{l+1} + \beta\left(\mathbf{A}_{l+1} - \mathbf{A}_l\right)$
10 $\quad l = l + 1$
11 **return** $\mathbf{A}_l$.

3. the computation of L and μ requires at most $O(R^3)$ arithmetic operations;

4. the computation of $\nabla f_\Omega(\mathbf{Y}_l)$ and the updates of $\mathbf{A}_l$ and $\mathbf{Y}_l$ require $O(PR)$ arithmetic operations.

4.3 Nonnegative Tensor Completion

At a high level, the update in the nonnegative case is given by

$$\mathbf{U}_{k+1}^{(i)} = \text{NMLSME}(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda). \tag{4.9}$$

In line 1 of Algorithm 3, matrix $\mathbf{W}_k^{(i)}$ is computed in a row-wise manner, with its j-th row, $\mathbf{W}_k^{(i)}(j,:)$, computed as follows

$$\mathbf{W}_k^{(i)}(j,:) := \left(\mathbf{M}_{(i)}(j,:) \circledast \mathbf{X}_{(i)}(j,:)\right)\mathbf{K}_k^{(i)}. \tag{4.10}$$

Note that for every inner iteration (indexed by l) in line 6 of Algorithm 3, we compute

$$\mathbf{Z}_{k,l}^{(i)} = \left(\mathbf{M}_{(i)} \circledast \left(\mathbf{Y}_{k,l}^{(i)}\mathbf{K}_k^{(i)T}\right)\right)\mathbf{K}_k^{(i)}. \tag{4.11}$$

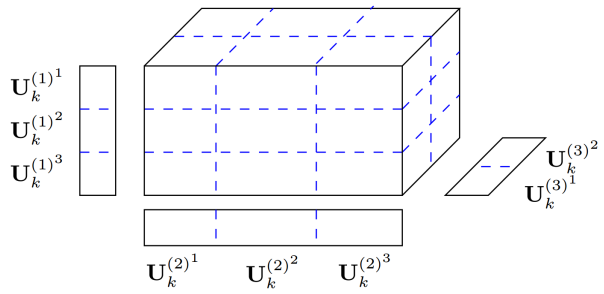

Figure 4.1: Tensor $\mathcal{X}$, factors $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, and $\mathbf{U}^{(3)}$, and their partitioning for $p_1 = p_2 = 3$ and $p_3 = 2$.

4.4 Alternating optimization for Tensor Completion

In Algorithm 4, we present the AO TC algorithm. We start from initial points $\mathbf{U}_0^{(i)}$, for $i = 1, \ldots, N$, and solve, in a circular manner, MLSME problems, based on the previous estimates. Similar to TD, we may perform an acceleration step after each AO outer iteration.

Algorithm 4: AO algorithm for NTC.

Input: $\mathcal{X}$, Ω, $\mathbf{U}_0^{(i)} \in \mathbb{U}^i$, $i = 1, \ldots, N$, λ.
1 $k = 0$
2 **while** (1) **do**
3 **for** $i = 1, 2, \ldots N$ **do**
4 $\mathbf{U}_{k+1}^{(i)} = \text{NMLSME}\left(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda\right)$
5 **if** (terminating condition is TRUE) **then** break; **endif**
6 $k = k + 1$
7 **return** $\mathbf{U}_k^{(i)}, i = 1, \ldots, N$.

4.5 Parallel Scheme

4.5.1 Topology preliminaries

We consider that we have available $p = \prod_{i=1}^{N} p_i$ processing elements. p_i corresponds to the number of processing units, for mode i. We describe in detail the implementation of the algorithms for the computation of the mode-N tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ onto an N-dimensional Cartesian processor space, whose processors are denoted as $p_{i_1,\ldots,i_N}$, with $i_j \in \{1, \ldots, p_j\}$

and $j \in \{1, \ldots, N\}$. We introduce certain partitionings of the factor matrices. We partition each factor matrix $\mathbf{U}^{(i)}$ into p_i block rows as

$$\mathbf{U}^{(i)} = \left[\begin{array}{ccc} \left(\mathbf{U}^{(i)^1}\right)^T & \cdots & \left(\mathbf{U}^{(i)^{p_i}}\right)^T \end{array} \right]^T, \tag{4.12}$$

with $\mathbf{U}^{ij} \in \mathbb{R}^{\frac{I_i}{p_i} \times R}$, for $j \in \mathbb{N}_{p_i}$.

Communication domains

We define certain communication domains (or processor groups) [31] over subsets of the p processors, which are used for the efficient collaborative implementation of specific computational tasks, as explained in detail below.

First, we define the $(N-1)$-dimensional groups of processors involving the $\prod_{k=1,k\neq i}^{N} p_k$ processors having the i-th index equal to j, i.e.,

$$\mathbb{P}_{i,j} := \{p_{k_1,k_2,\ldots,k_{i-1},j,k_{i+1},\ldots,k_N} : k_l \in \mathbb{N}_{p_l}, l \neq i\},\ i \in \mathbb{N}_N,\ j \in \mathbb{N}_{p_i}. \tag{4.13}$$

These processor groups form hyperlayers in the processor space and are used for the collaborative update of $\mathbf{U}_k^{(i)^j}$.

We also define the 1D processor groups involving the p_i processors that differ only at the i-th index, for $i \in \mathbb{N}_N$:

$$\mathbb{P}_{k_1,\ldots,k_{i-1},:,k_{i+1},\ldots,k_N} := \{p_{k_1,\ldots,k_{i-1},j,k_{i+1},\ldots,k_N} : j = 1, \ldots, p_i\},\ k_j \in \mathbb{N}_{p_j}. \tag{4.14}$$

Each of these groups forms a mode-i fiber in the processor space and is used for the collaborative computation of $\mathbf{U}_{k+1}^{(i)T}\mathbf{U}_{k+1}^{(i)}$, for $i \in \mathbb{N}_N$.

4.5.2 Variable partitioning and data allocation

We describe in detail the implementation of the NTC algorithm for the decomposition of a mode-N tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ on an N-dimensional Cartesian processor space, whose processors are denoted as $p_{i_1,\ldots,i_N}$, with $i_j \in \mathbb{N}_{p_j}$ and $j \in \mathbb{N}_N$.

At first, we introduce certain partitionings of the factor matrices and the tensor $\mathcal{X}$.

The collaborative update of $\mathbf{U}_k^{(i)^j}$ on $\mathbb{P}_{i,j}$ is achieved as follows:

1. Term $\mathbf{W}_k^{(i)^j}$ is computed in a collaborate manner on $\mathbb{P}_{i,j}$ and scattered among the processors of the group via a reduce-scatter operation.

2. Before the execution of the while loop of Algorithm 3, we must compute the global parameter $L = \max\left(\text{eig}\left(\mathbf{K}_k^{(i)T}\mathbf{K}_k^{(i)}\right)\right)$.

3. During each iteration of the while loop, indexed by l (see line 6 of Algorithm 3, we compute matrix $\mathbf{Z}_{k,l}^{(i)^j}$ according to (4.11). Each processor in the group $\mathbb{P}_{i,j}$ computes its contribution to $\mathbf{Z}_{k,l}^{(i)^j}$ in a row-wise fashion. By a reduce-scatter operation over $\mathbb{P}_{i,j}$, all processors in the group learn the appropriate rows of $\mathbf{Z}_{k,l}^{(i)^j}$ and perform an accelerated gradient step. An all-gather operation over $\mathbb{P}_{i,j}$ follows, and thus all processors in the group learn $\mathbf{Y}_{k,l+1}^{(i)^j}$ and become ready for the next iteration of the while loop.

4. After the end of Algorithm 3, the updated parts of $\mathbf{U}_k^{(i)^j}$ are all-gathered at all processors of the group $\mathbb{P}_{i,j}$, so that *all* processors in the group learn the updated factor $\mathbf{U}_{k+1}^{(i)^j}$.

5. By applying an all-reduce operation to $\left(\mathbf{U}_{k+1}^{(i)^j}\right)^T \mathbf{U}_{k+1}^{(i)^j}$, for $j \in \mathbb{N}_{p_i}$, on each of the mode-i 1D processor groups, *all* processors learn $\mathbf{U}_{k+1}^{(i)T}\mathbf{U}_{k+1}^{(i)}$. Thus, all processors can compute $\mathbf{K}_k^{(i+1)T}\mathbf{K}_k^{(i+1)}$.

4.6 Numerical Experiments

Next, we present results for the nonnegative tensor completion problem. In the cases where the tensor dimensions differ significantly over the modes the processor grid resembles the tensor, that is, we assign more processors along the modes with the largest dimensions.

We start with the nonnegative case with synthetic and real-world data. In Fig. 4.2 (left), we plot the execution time for a synthetic tensor of size $9,200 \times 9,200 \times 9,200$ with $8,000,000$ non-zero entries (99.999% sparsity) and $R = 15, 50$. In Figs. 4.3 (right) and 4.4 (left), we plot the execution times for synthetic tensors with size $71,567 \times 65,133 \times 171$ and $800,000 \times 1000 \times 1000$, respectively, both having $8,000,044$ non-zero entries (99.999% sparsity), and $R = 15, 50$.

In Fig. 4.5 (right), we consider the real-world nonnegative sparse dataset `Chicago Crime`, which concerns crime reports in the city of Chicago starting from January 1st 2001 up to December 11th, 2017 [32]. Data are arranged in a 4–th order tensor $\mathbf{X} \in \mathbb{R}_+^{6,186\times 24\times 77\times 32}$ with $5,330,673$ non-zeros. The modes correspond to (day, hour, community, crimetype), where 'community' is one of the communities of Chicago, and the non-zeros

represent the number of reports of a specific type of crime. In this experiment, we set $R = 10, 30$.

4.6.1 Comments

In Figure 4.4, we are noticing that for $R = 10$ and for $p = 2, 8, 64, 144$, the speedup achieved is above the linear speedup. This could be attributed to one of the dimensions being large, which makes memory access operations very costly in the serial case.

4.6.2 Speedup and time plots

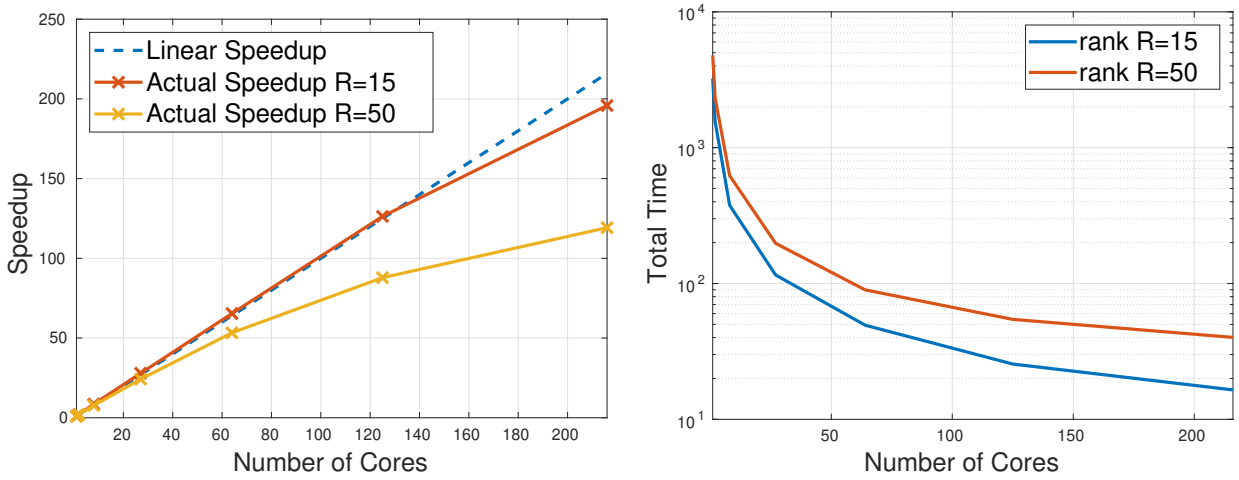

Figure 4.2: Speedup plot (left) and execution time in sec (right) for a synthetic nonnegative tensor of dimensions $9,200 \times 9,200 \times 9,200$ with $8,000,000$ elements $(99,99\%$ sparsity) and $p = 1, 2, 8, 27, 64, 125, 216$ cores.

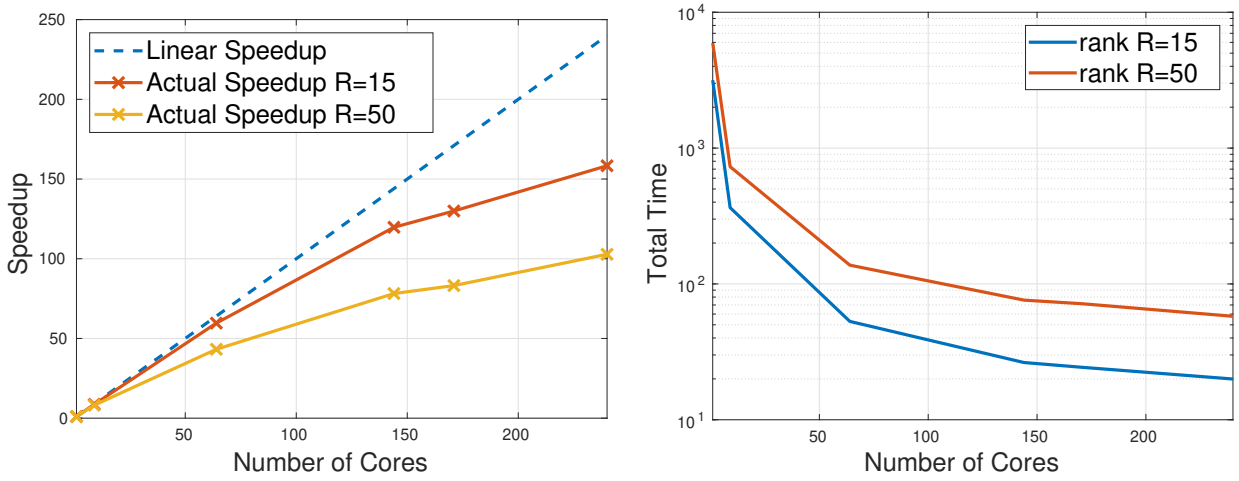

Figure 4.3: Speedup plot (left) and execution time in sec (right) for a synthetic nonnegative tensor of dimensions $71,567 \times 65,133 \times 171$ with $8,000,044$ elements ($99,99\%$ sparsity) and $p = 1, 9, 64, 144, 171, 240$ cores.

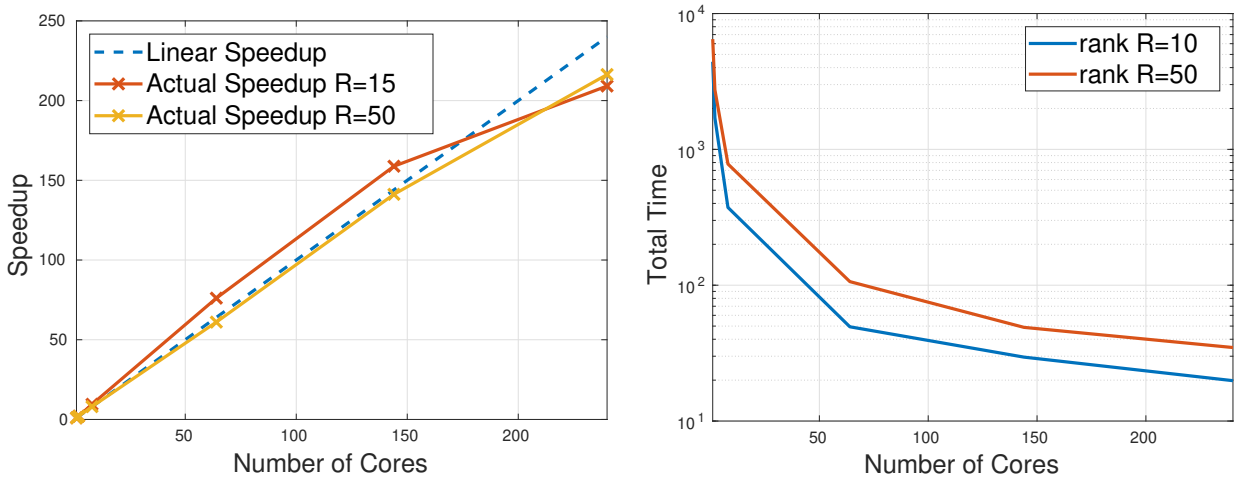

Figure 4.4: Speedup plot (left) and execution time in sec (right) for a synthetic nonnegative tensor of dimensions $800,000 \times 1,000 \times 1,000$ with $8,000,044$ elements ($99,99\%$ sparsity) and $p = 1, 2, 8, 64, 144, 240$ cores.

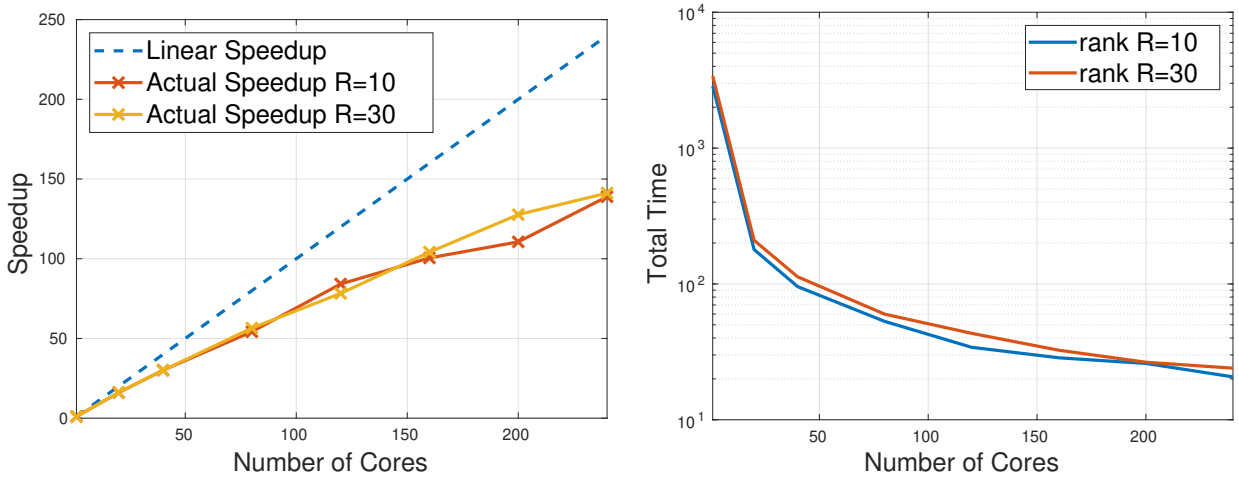

Figure 4.5: Speedup plot (left) and execution time in sec (right) for a tensor, formed from the Chicago Crime dataset, with dimensions $6,186 \times 24 \times 77 \times 32$ with $5,330,673$ elements ($68,62\%$ sparsity) and $p = 1, 20, 40, 80, 120, 160, 200, 240$ cores.

Chapter 5

Tensor Completion with Smoothing Constraints

We are now focusing on the CPD tensor completion problem that is augmented with smoothness regularization

$$\min f_{\Omega}\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) + \lambda \sum_{i=1}^{N} \|\mathbf{U}^{(i)}\|_F^2 + \sum_{i=1}^{N} \mu_i \|\mathbf{T}_i \mathbf{U}^{(i)}\|_F^2. \tag{5.1}$$

$\mathbf{T}_i \in \mathbb{R}^{(I_i-1)\times I_i}$ is a smoothness promoting matrix. We define it as $\mathbf{T}_i(j,j) = 1$ and $\mathbf{T}_i(j,j+1) = -1$, with the rest of the elements being equal to zero. We redefine $f_{\Omega}\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right)$ for this problem as

$$f_{\Omega}\left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{M} \left\| \mathcal{M} \circledast \left(\boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\right) \right\|_F^2, \tag{5.2}$$

where M is the number of available data points that we use for training and $\mathcal{M}$ is defined as in chapter 4. The authors in [24] use the cost function in (5.1) and try to approximate any arbitrary function, using the CP decomposition. The method is referred to as Canonical System Identification (CSID). They solve this problem using an alternating optimization method. In addition, for each factor, the update is performed in a row–wise manner. Let $\mathbf{u}_k^{(i)}$ denote the k–th row of the i–th factor, $\mathbf{m}_{(i)_k}$ the k–th column of the matricization of $\mathcal{M}$ with respect to the i–th mode, $\mathbf{x}_{(i)_k}$ the k–th column of the matricization of $\mathcal{X}$ with respect to the i–th mode and $\mathbf{K}^{(k)}$ is defined as in chapter 4. For each row, the following problem is being solved

$$\min_{\mathbf{u}_k^{(i)}} \frac{1}{M} \|\mathrm{diag}(\mathbf{m}_{(i)_k})(\mathbf{x}_{(i)_k} - \mathbf{K}^{(k)}\mathbf{u}_k^{(i)})\|_2^2 + \lambda \|\mathbf{u}_k^{(i)}\|_2^2 + \mu_k \|\mathbf{u}_k^{(i-1)} - \mathbf{u}_k^{(i)}\|_2^2 + \mu_k \|\mathbf{u}_k^{(i+1)} - \mathbf{u}_k^{(i)}\|_2^2. \tag{5.3}$$

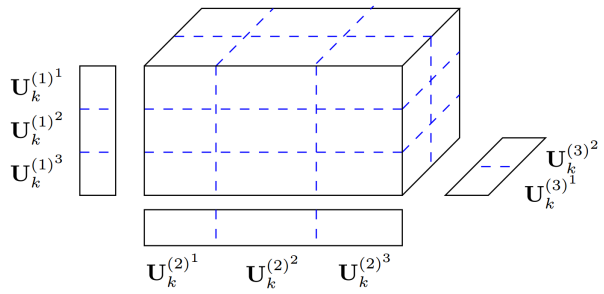

Figure 5.1: Tensor $\mathcal{X}$, factors $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, and $\mathbf{U}^{(3)}$, and their partitioning for $p_1 = p_2 = 3$ and $p_3 = 2$.

The solution for $\mathbf{u}_i^k$ is given by

$$\mathbf{u}_k^{(i)} = \begin{cases} (\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{K}^{(k)} + (\lambda+\mu_k)\mathbf{I})^{-1}(\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{x}_{(i)_k} - \mu_k(\mathbf{u}_k^{(i+1)})), & \text{if } k = 1 \\ (\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{K}^{(k)} + (\lambda+\mu_k)\mathbf{I})^{-1}(\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{x}_{(i)_k} - \mu_k(\mathbf{u}_k^{(i-1)})), & \text{if } k = I_i \\ (\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{K}^{(k)} + (\lambda+2\mu_k)\mathbf{I})^{-1}(\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{x}_{(i)_k} - \mu_k(\mathbf{u}_k^{(i-1)} + \mathbf{u}_k^{(i+1)})), & \text{else} \end{cases} \tag{5.4}$$

We notice that for the update of the i-th row we need the previous updated row, as well as the next one. The method has also been used in [33] for the problem of self–interference cancellation. The results are comparable to the state of the art algorithms utilizing neural networks.

5.1 Parallel Scheme

5.1.1 Topology preliminaries

We consider that we have available $p = \prod_{i=1}^{N} p_i$ processing elements. p_i corresponds to the number of processing units, for mode i. We describe in detail the implementation of the algorithms for the computation of the mode-N tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ onto an N-dimensional Cartesian processor space, whose processors are denoted as $p_{i_1,\ldots,i_N}$, with $i_j \in \{1,\ldots,p_j\}$ and $j \in \{1,\ldots,N\}$. We introduce certain partitionings of the factor matrices. We partition each factor matrix $\mathbf{U}^{(i)}$ into p_i block rows as

$$\mathbf{U}^{(i)} = \begin{bmatrix} \left(\mathbf{U}^{(i)^1}\right)^T & \cdots & \left(\mathbf{U}^{(i)^{p_i}}\right)^T \end{bmatrix}^T, \tag{5.5}$$

with $\mathbf{U}^{ij} \in \mathbb{R}^{\frac{I_i}{p_i} \times R}$, for $j \in \mathbb{N}_{p_i}$.

Communication Domains

As in chapter 4, we define the one–dimensional groups $\mathbb{P}_{k_1,\ldots,k_{i-1},:,k_{i+1},\ldots,k_N}$ and the $(N-1)$–dimensional groups $\mathbb{P}_{i,j}$. The latter are used for the collaborative computation of $\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{K}^{(k)}$ and $\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{x}_{(i)_k}$, while the former are used for send and receive operations needed in the algorithm.

5.1.2 Parallel Algorithm for Unconstrained Tensor Completion with Smoothing Constraints

There are no dependencies for the terms

$$(\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{K}^{(k)} + (\lambda + 2\mu_k)\mathbf{I})^{-1}$$

and

$$\mathbf{K}^{(k)^T}\text{diag}^2(\mathbf{m}_{(i)_k})\mathbf{x}_{(i)_k}.$$

We can exploit it when trying to derive a parallel scheme. Before we start a factor update, we suggest to first create the non-dependent terms and store them. We refer to this initial phase as *phase 1* of the algorithm.

We refer to the calculation of the updated rows and the propagation of the results as *phase 2*. Without loss of generality, let us consider the update of $\mathbf{U}^{(1)}$. Initially, the processes that contain $\mathbf{U}^{(1)^1}$ will start updating their rows, while the other processes will remain inactive. Once the computations are complete, these processes will have to send the last row of $\mathbf{U}^{(1)^1}$ to the processes that have $\mathbf{U}^{(1)^2}$, so they can start their respective computations. Once all lines of $\mathbf{U}^{(1)^2}$ are computed, the last row of this matrix will be sent to the processes that contain $\mathbf{U}^{(1)^3}$, and so on, until the whole factor has been updated. We note here, that before entering phase 2, all processes that have $\mathbf{U}^{(1)^p}$, where $p \in \{2,\ldots,p_1\}$, send the first row of $\mathbf{U}^{(1)^p}$ to the processes that have $\mathbf{U}^{(1)^{p-1}}$, so they can complete their respective factor update. A pseudo–algorithm describing this procedure is shown in Algorithm 5.

Table 5.1: Processor grids used for the Uber Pickups dataset

Number of Processors	Grid Formation
1	$1 \times 1 \times 1 \times 1$
4	$1 \times 1 \times 2 \times 2$
16	$1 \times 1 \times 4 \times 4$

Number of Processors	Grid Formation
64	$2 \times 1 \times 4 \times 8$
128	$4 \times 1 \times 4 \times 8$
256	$8 \times 1 \times 4 \times 8$

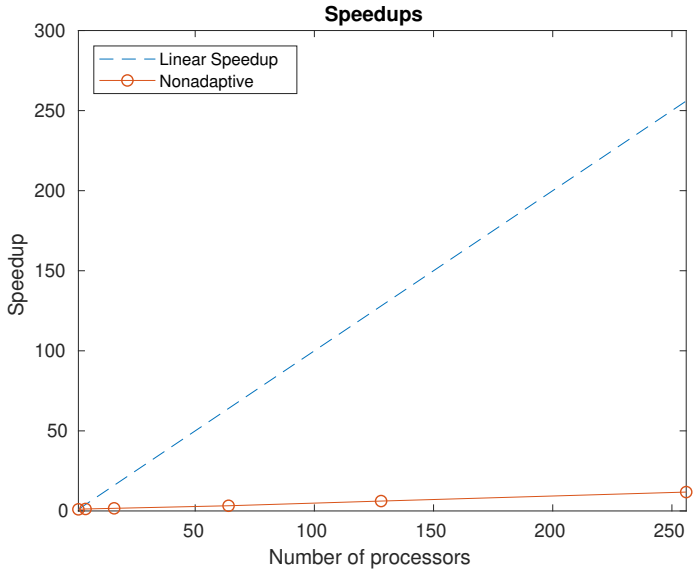

Figure 5.2: Speedup plot, for the Uber Pickups dataset, for a grid with the formations in table 5.1, for the initial partitioning scheme.

5.2 Adaptive Partitioning

5.2.1 Motivation

The nonzero elements of several real world datasets are nonuniformly distributed. As a result, if we partition the data into p processors, without taking the distribution of the nonzero elements into account, the speedup gain may not be substantial. Let us consider the dataset 'Uber Pickups'. The tensor formed is in $\mathbb{R}^{183\times24\times1,140\times1,717}$ and the distribution of the nonzeros across each mode appears in Figure 5.3. We consider the grid formations in Table 5.1 and set $R = 10$.

For the case where we do not take the distribution of the nonzeros into account, when creating the partitions for the tensor, the gained speedup is presented in Figure 5.2. We observe that the speedup gained is low.

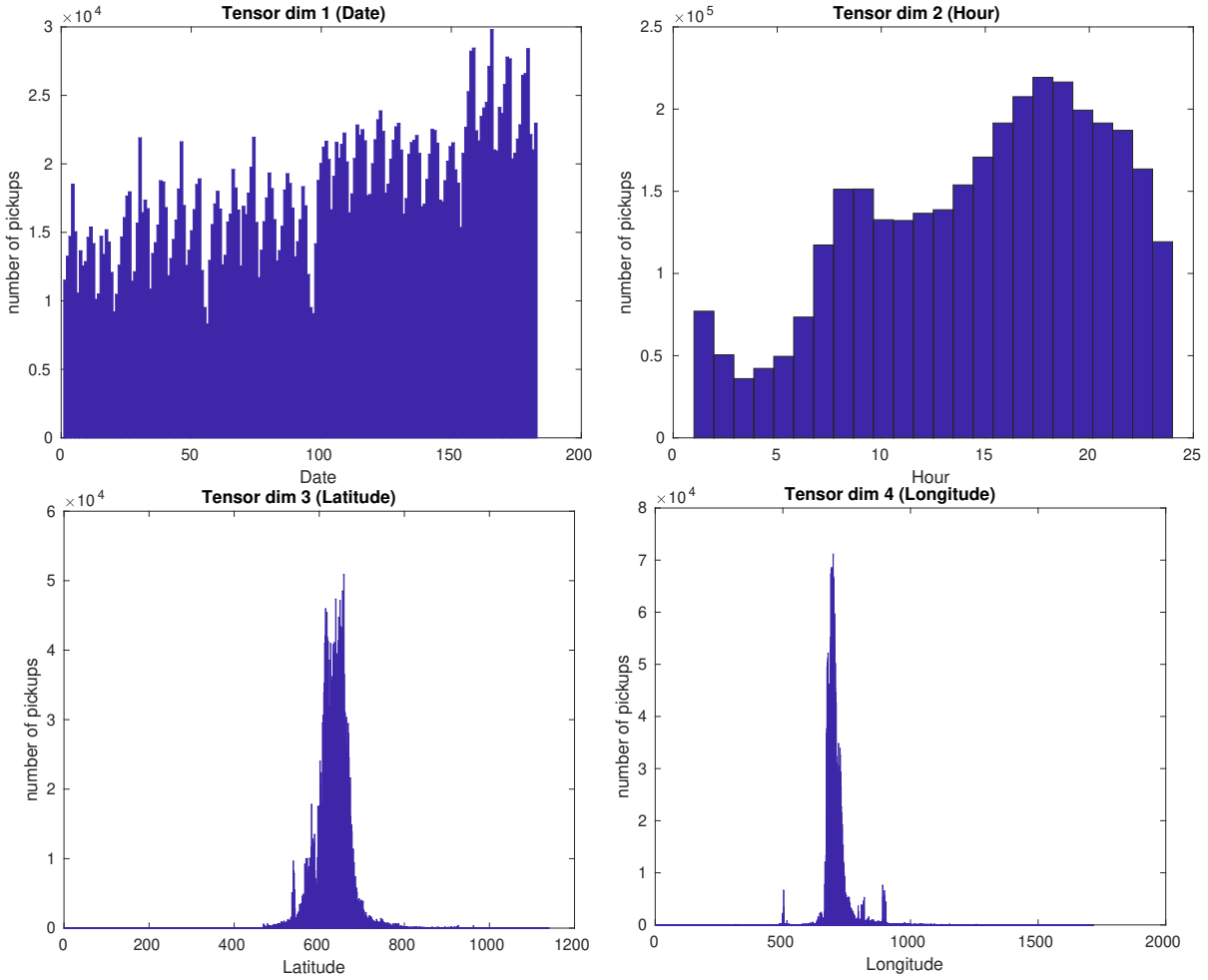

Figure 5.3: Distribution of nonzeros per dimension for the Uber Pickups dataset.

Algorithm 5: Algorithm for parallel Unconstrained Tensor Completion with Smoothing Constraints

Input: $[I_1, \ldots, I_N]$: Dimensions of the Tensor,
R: The rank of the decomposition,
D: The Dataset,
λ: Regularization parameter,
$[\mu_1, \ldots, \mu_N]$: Smoothing parameters for each dimension,
$[p_1, \ldots, p_N]$: Processes per mode,
M: Number of data points.

Result: $\boldsymbol{\mathcal{X}}$: The Tensor model that approaches the data,
$[\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]$: Rank-$R$ factors of CPD model.

```
D_local = Distribute_Dataset(D)
k = 0
repeat
    for i = 1, ..., N do
        Create inv_term_list, Kkyi_list
        Create y_i based on D_local
        for k = 1, ..., I_i / p_i do
            Create K_k based on D_local
            Kk_Kk = K^{k^T} K^k
            Kk_yi = K^{k^T} y_i
            All_reduce(Kk_Kk, layer_ comm[i])
            All_reduce(Kk_yi, layer_ comm[i])
            Kkyi_ list.append(Kk_Yi)
            Create inverse term and append to inv_term_list according to (5.4)
        if p_i > 1 then
            for p = p_i, p_i - 1, ..., 1 do
                process p sends u_1^(i) to process p - 1
                process p - 1 stores the received row
    for i = 1, ..., N do
        for p = 1, ..., p_i do
            for k = 1, ..., I_i / p_i do
                update u_k^(i) according to (5.4)
            if p_i > 1 and p ≠ p_i then
                process p sends u_k^(i) to process p + 1
until convergence is achieved, or maximum number of iterations has been reached
return [X, U^(1), ..., U^(N)]
```

Algorithm 6: Algorithm for adaptive partitioning

Input: N: Order of the tensor,
$[I_1, \ldots, I_N]$: Dimensions of the Tensor,
$[p_1, \ldots, p_N]$: The processors assigned to each mode,
nnz: The number of nonzeros,
$NNZ\_mode[][]$: Each cell contain the number of nonzeros for an index in a specific mode,
$thres\_error$: Tolerance for the number of nonzeros,

Result: $local\_dims$: The dimensions of the local tensor that each processor has,
$local\_true\_nnz$: The number of non–zero elements each processor has.

1 $MasterID = 1$
2 $local\_dims = cell(N, 1)$
3 $local\_optimal\_nnz = cell(N, 1)$
4 $local\_true\_nnz = cell(N, 1)$
5 **for** $i = 1, \ldots, N$ **do**
6 **if** $p_i > 1$ **then**
7 $isDone = 0$
8 $true\_threshold\_error = thres\_error$
9 **while** $isDone == 0$ **do**
10 **for** $p = 1, \ldots, p_i$ **do**
11 $local\_dims\{i\}(p) = \left\lfloor \frac{I_i}{p_i} \right\rfloor + (p == MasterID) * (I_i \ \% \ p_i)$
12 $local\_optimal\_nnz\{i\}(p) = \left\lfloor \frac{nnz}{p_i} \right\rfloor + (p == MasterID) * (nnz \ \% \ p_i)$
13 $threshold\_err\_i = \left\lceil true\_threshold\_error * \frac{nnz}{p_i} \right\rceil$
14 $pivot = 1$
15 **for** $p = 1, \ldots, p_i - 1$ **do**
16 $isBalanced = 0$
17 $nnz\_sum = 0$
18 **if** $pivot + local\_dims\{i\}(p) - 1 > I_i$ **then**
19 $isDone = 0$
20 **break**
21 **else**
22 $isDone = 1$

for $r = pivot, \ldots, pivot + local\_dims\{i\}(p) - 1$ **do**
    $nnz\_sum = nnz\_sum + NNZ\_mode\{i\}(r)$
while $isBalanced == 0$ **do**
    $error = nnz\_sum - local\_optimal\_nnz\{i\}(p)$;
    if $(error > threshold\_err\_i)$ & $(local\_dims\{i\}(p) > 0)$ & $(r > 0)$ **then**
        $local\_dims\{i\}(p) = local\_dims\{i\}(p) - 1$;
        $r = r - 1$;
        $nnz\_sum = nnz\_sum - NNZ\_mode\{i\}(r)$;
    else if $(error < -threshold\_err\_i)$ & $(local\_dims\{i\}(p) > 0)$ & $(r < I_i)$ **then**
        $local\_dims\{i\}(p) = local\_dims\{i\}(p) + 1$;
        $r = r + 1$;
        $nnz\_sum = nnz\_sum + NNZ\_mode\{i\}(r)$;
    else if $local\_dims\{i\}(p) \leq 0$ **then**
        $isDone = 0$;
        break;
    else
        $isBalanced = 1$;
$pivot = pivot + local\_dims\{i\}(p)$;
$local\_true\_nnz\{i\}(p) = nnz\_sum$;
if $(isDone == 1)$ & $(I_i - pivot + 1 > 0)$ **then**
    $local\_dims\{i\}(p+1) = I_i - pivot + 1$;
    $local\_true\_nnz\{i\}(p+1) = nnz - sum(local\_true\_nnz\{i\}(1 : p))$;
else
    $isDone = 0$;
    $true\_threshold\_error = true\_threshold\_error + threshold\_error$;
else
    $local\_dims\{i\}(1) = I_i$;
    $local\_optimal\_nnz\{i\}(1) = nnz$;
return $[\boldsymbol{\mathcal{X}}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]$.

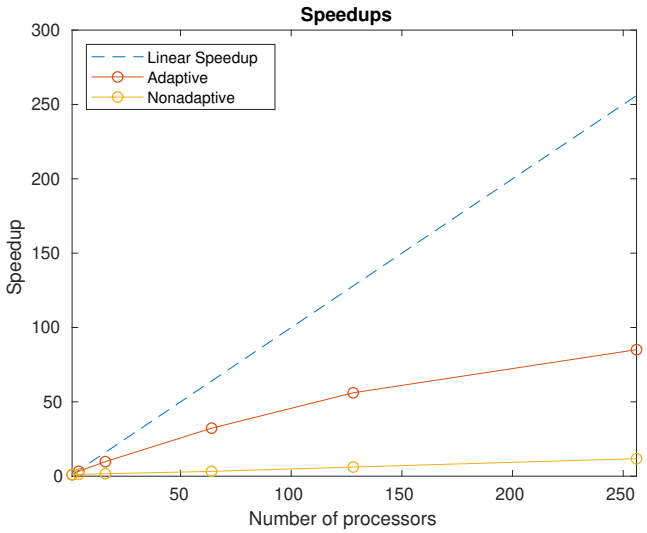

Figure 5.4: Speedup plot, for both partitioning methods, for a grid with the formations in Table 5.1.

5.2.2 The algorithm in a nutshell

The goal of the algorithm is, given the tolerance parameter *thres_error* and the optimal number of nonzeros for each processor, to readjust the dimensions for their local tensor, in order to have a number of nonzeros as close to the optimal number as possible. There may be cases where, given a threshold parameter, no possible partitioning can be created. In these cases, we double the threshold parameter and we repeat the algorithm. The algorithm is presented in Algorithm 6.

5.2.3 Result on Uber Pickups dataset

We present the results after using Algorithm 5 for the partitioning procedure, in Figure 5.4. We set *thres_error* equal to 0.1. We see that there is a substantial improvement in the performance over our initial algorithm.

5.3 Experiments on real world datasets

For all datasets considered, we will test for ranks $R = 10$ and $R = 50$. We also set *thres_error* to 0.1. We also tested for values of *thres_error* equal to 0.2 and 0.05, but those cases had worse results.

5.3.1 Chicago Crime

The tensor that is formed from this dataset is in $\mathbb{R}^{6,186\times24\times77\times32}$. The number of nonzeros is $5,330,673$. The distribution of the nonzeros is presented in Figure 5.5, while the grids that were used are presented in Table 5.2.

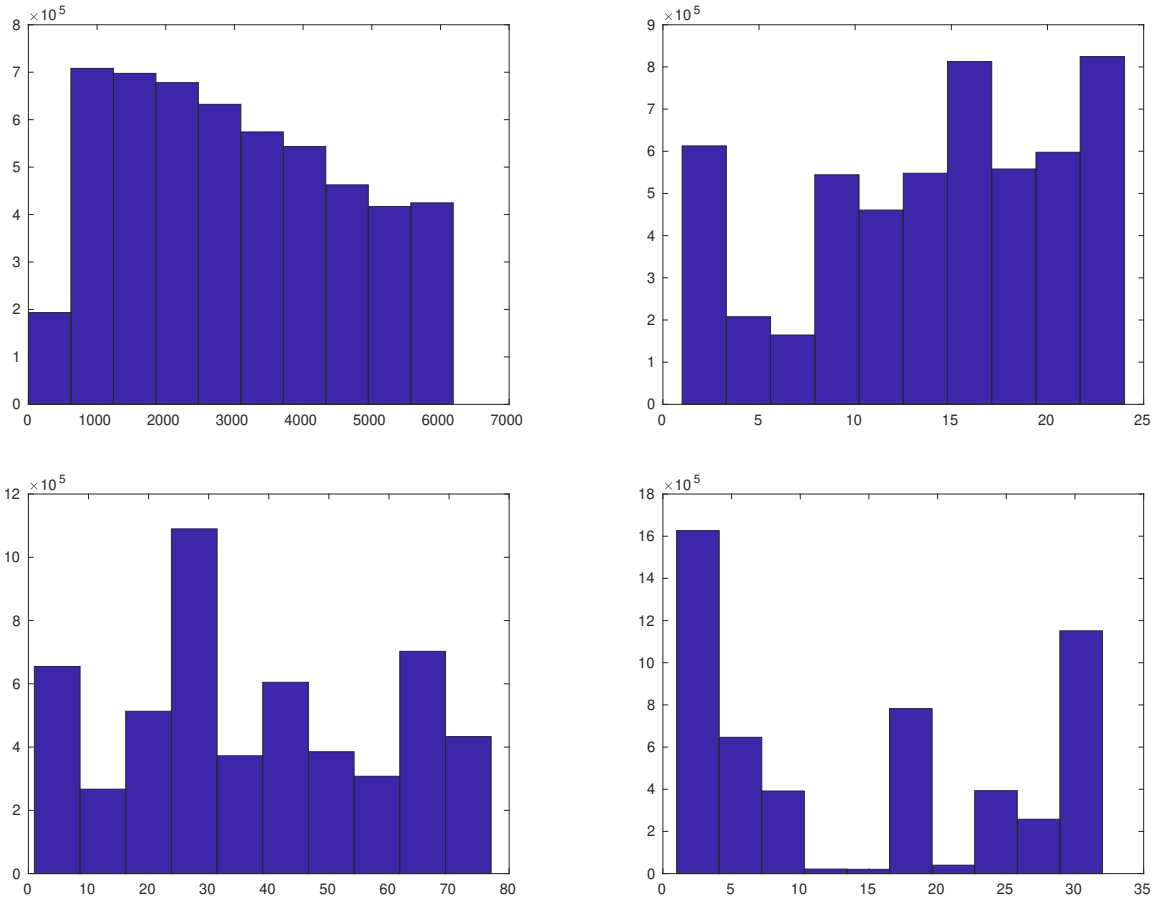

Figure 5.5: Distribution of nonzeros per dimension. **Upper left:** First Dimension, **Upper right:** Second Dimension, **Lower left:** Third Dimension, **Lower right:** Fourth Dimension.

Table 5.2: Processor grids used for the Chicago Crime dataset

Number of Processors	Grid Formation
1	$1 \times 1 \times 1 \times 1$
4	$4 \times 1 \times 1 \times 1$
8	$8 \times 1 \times 1 \times 1$
16	$16 \times 1 \times 1 \times 1$

Number of Processors	Grid Formation
32	$32 \times 1 \times 1 \times 1$
64	$64 \times 1 \times 1 \times 1$
128	$64 \times 1 \times 2 \times 1$
256	$64 \times 1 \times 2 \times 2$

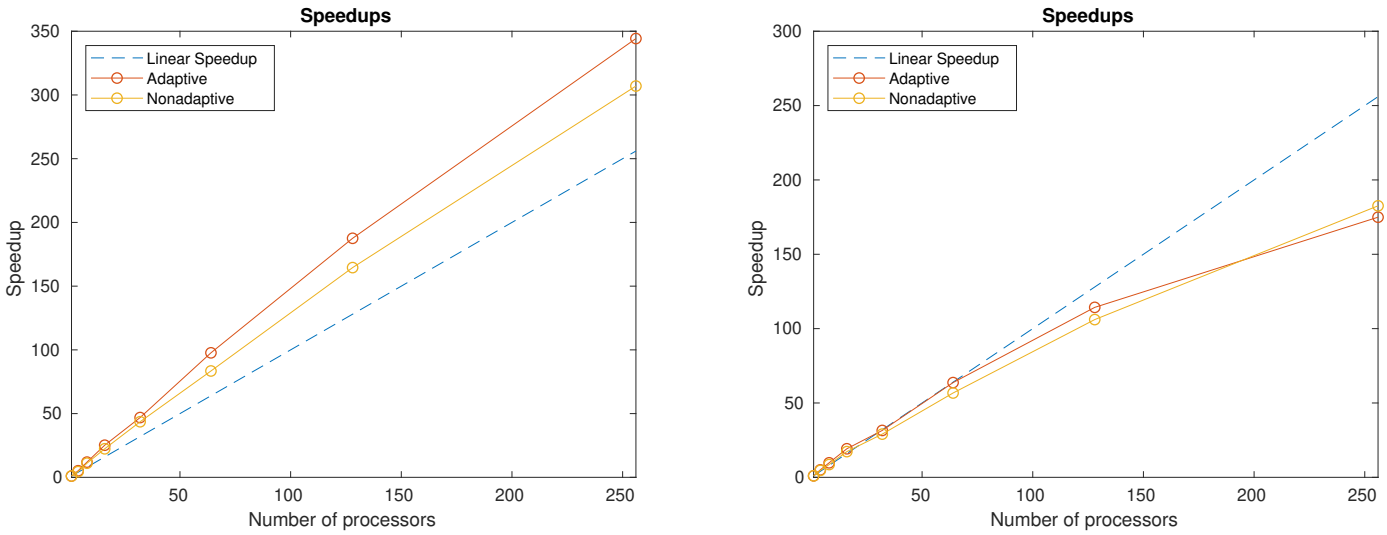

Figure 5.6: Speedup plots, for both partitioning methods, for a grid with the formations in Table 5.2, for the Chicago Crime dataset. Left is for $R = 10$, right is for $R = 50$.

5.3.2 Uber Pickups

The tensor that is formed from this dataset is in $\mathbb{R}^{183\times24\times1,140\times1,717}$. The number of nonzeros is $3,309,490$. The distribution of the nonzeros is presented in Figure 5.3, while the grids that were used are presented in Table 5.1.

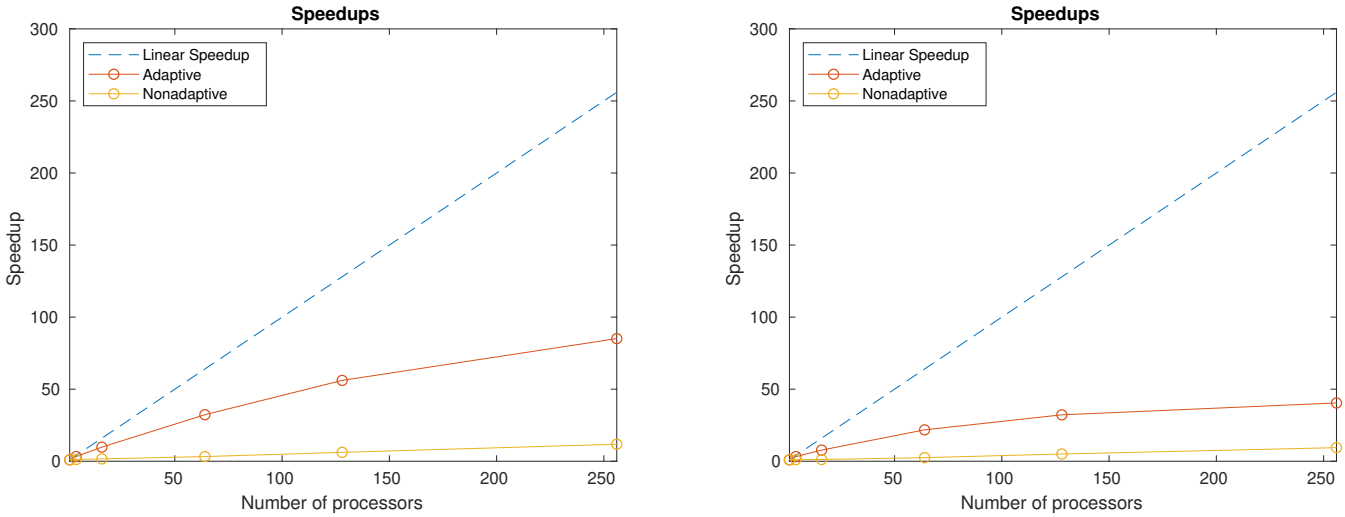

Figure 5.7: Speedup plot, for both partitioning methods, for a grid with the formations in Table 5.1, for the Uber Pickups dataset. Left is for $R = 10$, right is for $R = 50$.

5.3.3 Nips Publications

The tensor that is formed from this dataset is in $\mathbb{R}^{2,482\times 2,862\times 14,036\times 17}$. The number of nonzeros is $3,101,609$. The distribution of the nonzeros is presented in Figure 5.9, while the grids that were used are presented in Table 5.3.

Number of Processors	Grid Formation
1	$1 \times 1 \times 1 \times 1$
4	$1 \times 1 \times 4 \times 1$
8	$1 \times 1 \times 8 \times 1$
16	$1 \times 1 \times 16 \times 1$
32	$1 \times 2 \times 16 \times 1$
64	$2 \times 2 \times 16 \times 1$
128	$2 \times 4 \times 16 \times 1$
256	$4 \times 4 \times 16 \times 1$

Table 5.3: Processor grids used for the Nips Publications dataset.

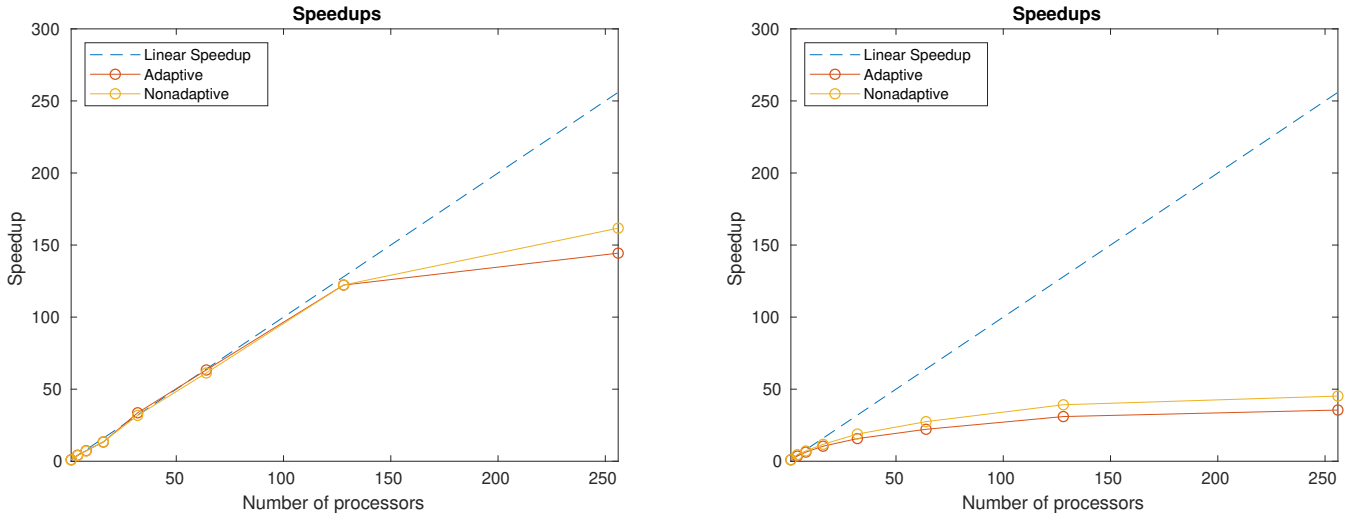

Figure 5.8: Speedup plot, for both partitioning methods, for a grid with the formations in Table 5.3, for the Nips Publications dataset. Left is for $R = 10$, right is for $R = 50$.

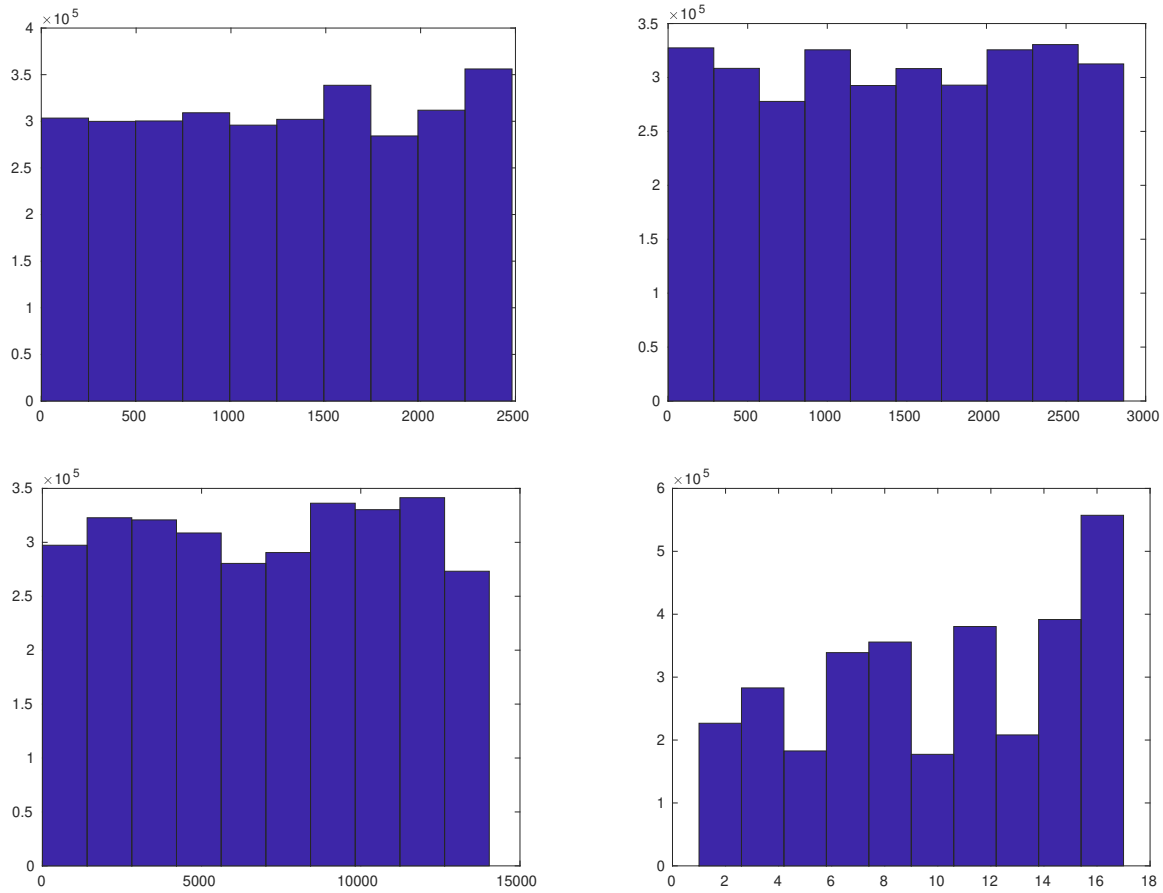

Figure 5.9: Distribution of nonzeros per dimension. **Upper left:** First Dimension, **Upper right:** Second Dimension, **Lower left:** Third Dimension, **Lower right:** Fourth Dimension.

Chapter 6

Discussion and Future Work

6.1 Conclusions

We considered tensor completion problems. We first studied nonnegative tensor completion. We expanded on the work in [34], by solving the problem in an N–dimensional grid, instead of a linear array. We note that the speedup gained in our experiments is substantial. Next, we studied unconstrained tensor completion problems with smoothing constraints. We provided a distributed solution for the problem and tested the speedup gain for our method. We developed a method that creates a partitioning for the tensor, based on the distribution of the nonzeros. This makes it possible to attain larger speedups, in cases where the initial speedup would be low.

6.2 Future Work

We conclude this thesis by presenting possible future extensions of this work.

6.2.1 Distributed Nonnegative Tensor Completion with Smoothing Constraints

We examined a distributed algorithm for unconstrained problems with smoothing constraints. A development of a similar algorithm for Nonnegative Tensor Completion with Smoothing Constraints could be a possible topic of interest.

6.2.2 Tensor Completion with other possible constraints

Another topic of interest could be the studying of algorithms for tensor completion for constraints other than nonnegativity and smoothness. Some examples include orthogonality constraints and sparsity constraints.

6.2.3 Distributed Algorithms for Nonnegative Tensor Completion with other tensor models

In this thesis, we considered the PARAFAC model for the task of tensor completion. An additional topic of interest would be the development of distributed algorithms for Nonnegative Tensor Completion that utilize other models (for example, Nonnegative Tucker Decomposition).

Bibliography

[1] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[2] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Wiley-Interscience, 2008.

[3] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Wiley, 2009.

[4] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] I. Oseledets, "Tensor-train decomposition," *SIAM J. Scientific Computing*, vol. 33, pp. 2295–2317, 01 2011.

[6] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[7] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion." *IEEE Journal on Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 586–600, 2015.

[8] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[9] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, accepted for publication, May 2016.

[10] P. A. Karakasis, C. Kolomvakis, G. Lourakis, G. Lykoudis, I. M. Papagiannakos, I. Siaminou, C. Tsalidis, and A. P. Liavas, "Partensor," *Tensors for Data Processing: Theory, Methods, and Applications*, pp. 66–90, 2021.

[11] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 284–295, 2015.

[12] C. Battaglino, G. Ballard, and T. G. Kolda, "A practical randomized cp tensor decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 39, no. 2, pp. 876–901, 2018.

[13] X. Fu, S. Ibrahim, H.-T. Wai, C. Gao, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *IEEE Transactions on Signal Processing*, 2020.

[14] I. Siaminou and A. P. Liavas, "An accelerated stochastic gradient for canonical polyadic decomposition," Eusipco 2021.

[15] I. Siaminou, I. M. Papagiannakos, C. Kolomvakis, and A. P. Liavas, "Accelerated stochastic gradient for nonnegative tensor completion and parallel implementation," Eusipco 2021.

[16] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Computing*, 2015.

[17] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[18] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 77.

[19] T. Yokota, R. Zdunek, A. Cichocki, and Y. Yamashita, "Smooth nonnegative matrix and tensor factorizations for robust multi-way data analysis," *Signal Processing*, vol. 113, pp. 234–249, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168415000614

[20] M. Imaizumi and K. Hayashi, "Tensor decomposition with smoothness," in *ICML*, 2017.

[21] F. Ma, F. Yang, and Y. Wang, "Low-rank tensor decomposition with smooth and sparse regularization for hyperspectral and multispectral data fusion," *IEEE Access*, vol. PP, pp. 1–1, 07 2020.

[22] Y.-B. Zheng, T.-Z. Huang, T.-Y. Ji, X.-L. Zhao, T.-X. Jiang, and T.-H. Ma, "Low-rank tensor completion via smooth matrix factorization," *Applied Mathematical Modelling*, vol. 70, pp. 677–695, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X19300782

[23] T. Yokota, Q. Zhao, and A. Cichocki, "Smooth parafac decomposition for tensor completion," *IEEE Transactions on Signal Processing*, vol. 64, no. 20, pp. 5423–5436, 2016.

[24] N. Kargas and N. D. Sidiropoulos, "Nonlinear system identification via tensor completion," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 4420–4427. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/5868

[25] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Nenmf: An optimal gradient method for nonnegative matrix factorization," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.

[26] Y. Zhang, G. Zhou, Q. Zhao, A. Cichocki, and X. Wang, "Fast nonnegative tensor factorization based on accelerated proximal gradient and low-rank approximation," *Neurocomputing*, vol. 198, no. Supplement C, pp. 148 – 154, 2016.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[28] Y. Nesterov, *Introductory lectures on convex optimization*. Kluwer Academic Publishers, 2004.

[29] B. O' Donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[30] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1126–1153, 2013.

[31] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Pearson, 2003.

[32] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/

[33] F. Jochems and A. Balatsoukas-Stimming, "Non-linear self-interference cancellation via tensor completion," *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pp. 905–909, 2020.

[34] G. Lourakis and A. P. Liavas, "Nesterov-based alternating optimization for nonnegative tensor completion: Algorithm and parallel implementation," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018, pp. 1–5.