Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Sihan Yue April 7, 2019

Smoothing Tensor Factorization on Spatio-Temporal Data

by

Sihan Yue

Dr. Joyce C. Ho
Adviser

Department of Mathematics, Honors Program

Dr. Joyce C. Ho

Adviser

Dr. Lars Ruthotto

Committee Member

Dr. Shomu Banerjee

Committee Member

2019

Smoothing Tensor Factorization on Spatio-Temporal Data

By

Sihan Yue

Dr. Joyce C. Ho

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Abstract

Smoothing Tensor Factorization on Spatio-Temporal Data
By Sihan Yue

As spatio-temporal data violates many assumptions required in traditional machine learning/ data mining algorithms, tensor factorization (TF) has often been adopted in analyzing such data. Yet, the non-smooth factors that TF outputs sometimes misrepresent the underlying structure of the data and hinder the interpretability without domain knowledge. With the goal of smoothing the factors, we proposed three approaches: i) adopting Tikhnov regularization to CP_OPT; ii) adopt CP_OPT_SMOOTH in ParCube; iii) ParCube with neighbor padding. In order to examine the performance of these algorithms, we performed numerical experiments on the New York Uber Pickups dataset provided by FROSTT. Our results show that i) CP_OPT_SMOOTH improves the smoothness and the runtime with certain cost of accuracy; ii) with CP_OPT_SMOOTH, CP_OPT can now be adopted in ParCube but with some sacrifice in accuracy; iii) neighbor padding improves the smoothness while maintaining high accuracy.

Smoothing Tensor Factorization on Spatio-Temporal Data

By

Sihan Yue

Dr. Joyce C. Ho

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spatio-temporal data, as indicated by its name, are data where space (e.g. latitude and longitude) and time attributes are ubiquitous. Massive amounts of spatio-temporal data are collected across many application domains including ride-sharing platforms for saving energy consumption and reducing traffic congestion, urban planning through a better understanding of real estate market analysis, and population health management to identify hot spot areas to deploy portable clinics to prevent widespread epidemics. These seemingly distinct applications have the same underlying theme, the need for automated discovery that centers around spatio-temporal-based pattern mining.Since spatio-temporal observations are highly correlated with time and location and therefore do not follow traditional assumptions of an independent and identically distributed variable, classic machine learning and data mining algorithms on spatio-temporal data usually do not yield fast and accurate results as they would on traditional data. Despite the complexity of the data itself, analyses

on spatio-temporal data often require domain-specific manual work, and are not computationally feasible for analyzing large-sized data [3, 19]. Some other problems that hamper the usage of classical machine learning algorithms include the variability in measurements with respect to length and frequency, and multi-sourced data that spans multiple sources of information and collected at multiple sites.

One common approach to represent spatio-temporal data is to use a data structure called tensor, which is a generalization of a matrix to multi-way data. A tensor is a natural representation for high-dimensional since it is powerful and flexible in allowing various succinct encodings of spatio-temporal data [5, 6, 14]. For example, ride-hailing demand can be represented as a three-way tensor where each element represents the number of requests for a specific location, the hour, and the day. Alternatively, the same information could be stored as a four-way tensor where the location is encoded using the latitude and longitude. Therefore, the tensor representation provides a sufficient representation for analyzing large-scale spatio-temporal data.

Tensor factorization is a common technique that provides a data-driven approach for automated discovery. Examples of tensor factorization methods for spatio-temporal analyses include detecting anomalous urban mobility patterns,

estimating path travel time, and forecasting points of interest in cities. The PARAFAC/CANDECOMP (CP) model is one of the most popular tensor analysis methods [2, 4]. CP decomposes a tensor into a sum of rank-one outer products which effectively represents the underlying data concepts. Its popularity owes to its intuitive output structure and uniqueness property that make the model reliable to interpret [6, 7]. Common algorithms to solve the CP model includes CP_ALS and CP_OPT. Although CP_ALS is easy to understand and implement, the algorithm is not stable since it is not guaranteed to converge to a solution where the objective function of CP ceases to decrease, not necessarily to a global minimum or even a stationary point [6]. CP_OPT is a gradient-based method. It achieves higher accuracies than CP_ALS at the cost of running three times slower than CP_ALS [10].

However, the resulting factors of the CP model are usually not smooth, which sometimes do not provide ideal information. For example, when analyzing a ride-hailing demand, we typically expect smooth transition in values as we move along the time coordinate and we expect close neighborhoods to have similar information as well. However, CP factorizations often yield results that have spikes. In order to obtain more accurate results, we would prefer the solution's components to be as smooth as possible. Earlier works focus on promoting

non-overlapping results for a specific, fixed tensor mode [18]. Yet, as such an objective is still limited by the need for prior domain knowledge or even an arbitrary choice of a tensor mode, recent works have started transitioning to focus on promoting multi-modal smooth decomposition results. A similar work, CP_ORTHO which focuses on promoting non-orthogonality for better interpretation on multi-modes, has also suggested a similar approach could be applied to promote smoothness [1].

1.1 Contributions

In this paper, we propose to modify known CP factorization algorithms to improve the smoothness of the decomposition factors. Our contributions are as folow:

- First, we propose to explore spatial and temporal smoothness using Tikhonov regularization on a stable and accurate algorithm CP_OPT. We name this algorithm CP_OPT_SMOOTH.
- Second, to scale to large datasets that may not fit into memory, we propose to replace the core decomposition of ParCube with CP_OPT_SMOOTH.
- Lastly, we propose a modification to the sampling technique used in Par-

Cube so that a smoother set of data will be sampled to provide more information of the underlying structure of the data.

The rest of the paper is organized as follows. In Chapter 2, we introduce the notations and the preliminaries necessary for easier understanding of our paper. We also introduce tensor factorization algorithms related to our approach and discuss their limitations. We then present our approaches to smooth decomposition factors in Chapter 3. Finally, we follow up with experiments on a real-life application in Chapter 4 and provide conclusions and future work directions in Chapter 5.

Chapter 2

Background

In this chapter, we first introduce the notations needed to quickly follow the algorithms in this paper in section 2.1. A brief introduction to tensors and common operators are provided in section 2.2. Section 2.3 offers an overview of matrix and tensor factorization, and section 2.4 reviews an efficient parallelizable method for tensor factorization (ParCube). Lastly, section 2.5 covers smoothing techniques.

2.1 Notation

Scalars are denoted by lowercase letters, x. Vectors are denoted by boldface lowercase letters, $\mathbf{x}$. Matrices are denoted by boldface capital letters, X. Higher-order tensors are denoted by boldface Euler script letters, $\mathscr{X}$. The i^{th} entry of a vector a is denoted by x_i, similarly, x_{ij} denotes element (i, j) of a matrix X, and x_{ijk} denotes element (i, j, k) of a third-order tensor $\mathscr{X}$. The i_{th} column of a

matrix X is denoted by x_i. Indices typically range from 1 to their capital version, e.g.,$i = 1, ..., I$. The n^{th} element in a sequence is denoted by a superscript in parentheses, e.g. $A^{(n)}$ denotes the nth matrix in a sequence.

2.2 Tensor and Common Operators

A tensor is a multidimensional or N-way array. In other words, a tensor is a generalization of matrix to multiway arrays. The order or mode of a tensor is the number of dimensions. An example of a third-order tensor is shown in Figure 2.1.

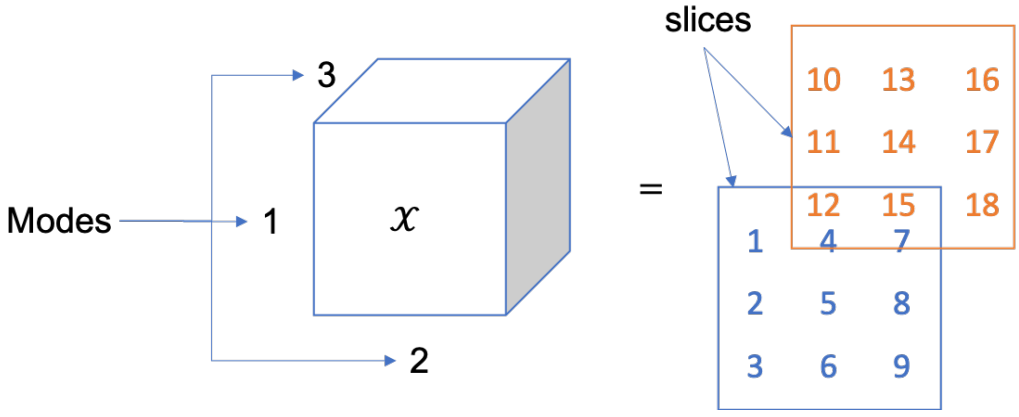

Figure 2.1: Example of a tensor
A third-order (or 3-mode) tensor of size $3 \times 3 \times 2$ with the tensor elements shown on the right side.

Common tensor operations are slices and matricization. *Slices* are two-

dimensional sections of a tensor, defined by fixing all but two modes. Matricization is the process of unfolding or flattening the tensor by reordering the elements of an N-way array into a matrix. For example, for a tensor $\mathcal{X}$, as shown in Figure 2.1, who has the two following slices ,

$$\mathcal{X}_{(1)} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}, \qquad \mathcal{X}_{(2)} = \begin{bmatrix} 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{bmatrix}$$

its corresponding mode-n unfoldings are:

$$\mathcal{X}_{(1)} = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 \\ 2 & 5 & 8 & 11 & 14 & 17 \\ 3 & 6 & 9 & 12 & 15 & 18 \end{bmatrix}$$

$$\mathcal{X}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 10 & 11 & 12 \\ 4 & 5 & 6 & 13 & 14 & 15 \\ 7 & 8 & 9 & 16 & 17 & 18 \end{bmatrix}$$

$$\mathcal{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{bmatrix}$$

The *norm* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the square root of the sum of the

squares of all its elements:

$$\|\mathcal{X}\|_F = \sqrt{\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N}^2}$$

$\mathcal{X}$ is an N-way *rank one* tensor if it can be written as the outer product of N vectors, $\mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)}$, where each element $x_{\vec{i}} = x_{i_1,i_2,\cdots,i_N} = a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)}$.

2.3 Matrix & Tensor Factorization

Matrix factorization (MF) is a common dimensionality reduction approach, which represents the original data using a lower dimensional latent space. A standard MF approach is to find two lower dimensional matrices that when multiplied together approximately produce the original matrix. The standard formulation for MF is as follows: given a $n \times m$ matrix $\mathbf{X}$, find matrices $\mathbf{W}$ and $\mathbf{H}$ of size $n \times r$ and $r \times m$ such that $\mathbf{X} \approx \mathbf{WH}$.

Tensor factorization (decomposition) is a natural extension of matrix factorization. The factorization of higher-order tensors have several advantages over matrix factorization as it utilizes information from the multi-way structure that is lost when modes are collapsed to use matrix factorization algorithms [9, 17], can identify components with very few observations [11], and has uniqueness of the optimal solution without imposing orthogonality and independence of

the factors. In the case of spatio-temporal data, the information on time and location can be easily lost when we collapse the modes to use matrix factorization. Using tensor factorization, on the other hand, will render a specific and explicit decomposition where both time and location, or even time, longitude and latitude, are straightforward for interpretation.

While a wide variety of decompositions are available [6, 11], we will focus on the CANDECOMP / PARAFAC (CP) model [2, 4]. The CP model, illustrated in Figure 2.2, approximates the original tensor $\mathscr{X}$ as a sum of R rank-one components

$$\begin{aligned}\mathscr{X} &\approx \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \ldots \circ \mathbf{a}_r^{(N)} \\ &= [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \ldots; \mathbf{A}^{(n)}]\!].\end{aligned}$$

Note that $[\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \ldots; \mathbf{A}^{(n)}]\!]$ is shorthand notation to describe the CP decomposition, where $\boldsymbol{\lambda}$ is a vector of the weights λ_r and $\mathbf{a}_r^{(n)}$ is the r^{th} column of $\mathbf{A}^{(n)}$. The standard formulation for finding the CP decomposition is posed as the following optimization problem:

$$\begin{aligned}\min_{\hat{\mathscr{X}}} \; & f = \frac{1}{2} ||\mathscr{X} - \hat{\mathscr{X}}||_F^2 \\ \text{s.t. } & \hat{\mathscr{X}} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \ldots; \mathbf{A}^{(n)}]\!]\end{aligned}$$

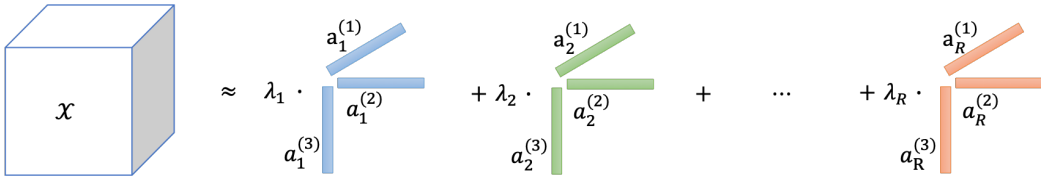

Figure 2.2: CANDECOMP/PARAFAC tensor decomposition

2.3.1 CP_ALS

The typical method for finding the CP components is alternating least squares (ALS) optimization, as proposed in the original CP papers [6, 11]. The premise is to iteratively optimize one factor matrix at a time while holding other modes fixed, rather than solving for $\mathbf{A}^{(1)}$ through $\mathbf{A}^{(N)}$ simultaneously. For example, in order to factor a three-way tensor, we start with an intial guess for the matrices **A, B**, and **C**, and then solve a least square problem for **A** while holding **B** and **C** fixed, then solve for **B** while fixing new **A** and **C**, and so on until the factors converge, as illustrated in Figure 2.3 .

Unfolded Tensor on the kth mode

$$\mathbf{T}_{(1)} = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T$$
$$\mathbf{T}_{(2)} = \mathbf{B}(\mathbf{C} \odot \mathbf{A})^T$$
$$\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T$$

$$\mathbf{A}^{k+1} = \underset{\widehat{\mathbf{A}} \in \mathbb{R}^{I \times R}}{\operatorname{argmin}} \|\mathbf{T}_{(1)} - \widehat{\mathbf{A}}(\mathbf{C}^k \odot \mathbf{B}^k)^{\mathrm{T}}\|_F^2,$$
$$\mathbf{B}^{k+1} = \underset{\widehat{\mathbf{B}} \in \mathbb{R}^{J \times R}}{\operatorname{argmin}} \|\mathbf{T}_{(2)} - \widehat{\mathbf{B}}(\mathbf{C}^k \odot \mathbf{A}^{k+1})^{\mathrm{T}}\|_F^2,$$
$$\mathbf{C}^{k+1} = \underset{\widehat{\mathbf{C}} \in \mathbb{R}^{K \times R}}{\operatorname{argmin}} \|\mathbf{T}_{(3)} - \widehat{\mathbf{C}}(\mathbf{B}^{k+1} \odot \mathbf{A}^{k+1})^{\mathrm{T}}\|_F^2$$

Figure 2.3: CP_ALS illustrated on a 3-way tensor

With all but one factor matrix fixed, the problem reduces to a linear least squares problem and has an exact solution. Thus, it is often the method of choice due to its speed and ease of implementation. Yet, CP_ALS often fails to obtain the underlying structure in the data, especially in the case of overfactoring. Since we cannot know the rank in advance, we often face the problem of overfactoring, i.e., computing CP when R is greater than the rank of the tensor [10].

2.3.2 CP_NMU

Nonnegative CP with multiplicative updates (NMU) is an extension of CP_ALS. CP_NMU differs from CP_ALS in that it imposes the constraint that the elements in each factor need to be nonnegative. This idea first arises in matrix factorization when Paatero and Tapper[12] and Lee and Seung[8] proposed us-

ing nonnegative matrix factorizations (NMF) for analyzing non-negative data, such as environmental models and grayscale images, because it is desirable for the decompositions to retain the nonnegative characteristics of the original data and thereby facilitate easier interpretation[6]. As tensor factorizations are generalizations of matrix factorizations, it is natural to extend NMF to tensor factorizations by simply including additional constraints of non-negativity.

$$\min_{\hat{\mathcal{X}}} f = \frac{1}{2}||\mathcal{X} - \hat{\mathcal{X}}||_F^2$$

$$\text{s.t. } \hat{\mathcal{X}} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \ldots; \mathbf{A}^{(n)}]\!]$$

$$\mathbf{A}^{(i)} \geq 0 \; \forall \; i = 1, 2, ..., n$$

Note that $\mathbf{A}^{(i)} \geq 0$ denotes that each element of $\mathbf{A}^{(i)}$ is nonnegative. An example of CP_NMU on a 3-way tensor is illustrated in Figure 2.4, where the parts underlined in red highlight the difference between CP_NMU and CP_ALS:

$$\min_{\mathbf{A} \geq 0} \|\mathbf{T}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T\|$$
$$\min_{\mathbf{B} \geq 0} \|\mathbf{T}_{(2)} - \mathbf{B}(\mathbf{C} \odot \mathbf{A})^T\|$$
$$\min_{\mathbf{C} \geq 0} \|\mathbf{T}_{(3)} - \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T\|$$

Figure 2.4: CP_ALS illustrated on a 3-way tensor

2.3.3 CP_OPT

In order to improve accuracy and stability, a gradient-based optimization approach for CP was proposed. The fundamental idea is to consider the CP objective function f as a mapping from the cross-product of N two-dimensional vector spaces to $\mathbb{R}$, and think of f as a scalar-valued function where the parameter vector $\mathbf{x}$ comprises the vectorized and stacked matrices $\mathbf{A}^{(1)}$ through $\mathbf{A}^{(N)}$,

$$x = \begin{bmatrix} \mathbf{a_1}^{(1)} \\ \vdots \\ \mathbf{a_R}^{(1)} \\ \vdots \\ \mathbf{a_1}^{(N)} \\ \vdots \\ \mathbf{a_R}^{(N)} \end{bmatrix}$$

Now, the gradient of f can be obtained by calculating the partial derivative with respect to each $\mathbf{a_r}^{(n)}$ for r = 1,...,R and n = 1,...,N with the following formula,

$$\frac{\partial f}{\partial \mathbf{a_r}^{(n)}} = -(\mathbf{Z} \bigtimes_{m=1, m\neq n}^{N} \mathbf{a_r}^{(m)}) + \sum_{l=1}^{R} \gamma_{\mathbf{rl}}{}^{(n)} \mathbf{a_l}^{(n)}$$

where ${\gamma_{\mathbf{rl}}}^{(n)}$ is defined as,

$$ {\gamma_{\mathbf{rl}}}^{(n)} = \prod_{m=1, m\neq n}^{N} {\mathbf{a_r}}^{(m)} {\mathbf{a_l}}^{(m)} $$

The objective function can now be optimized using any first-order optimization methods such as nonlinear conjugate gradient method or L-BFGS quasi-Newton method. Compared to CP_ALS, CP_OPT solves for all factor matrices simultaneously and the numerical results from the original paper show that this leads to increased accuracy in the case of overfactoring [10].

2.4 Parallelizable Tensor Factorization (ParCube)

ParCube is a fast and parallelizable method for speeding up tensor decompositions by leveraging random sampling techniques. The idea of the algorithm is to randomly under-sample a tensor multiple times, process the different samples in parallel and cleverly combine the results at the end to obtain high approximation accuracy at low complexity and main memory cost as showin in Figure 2.4.

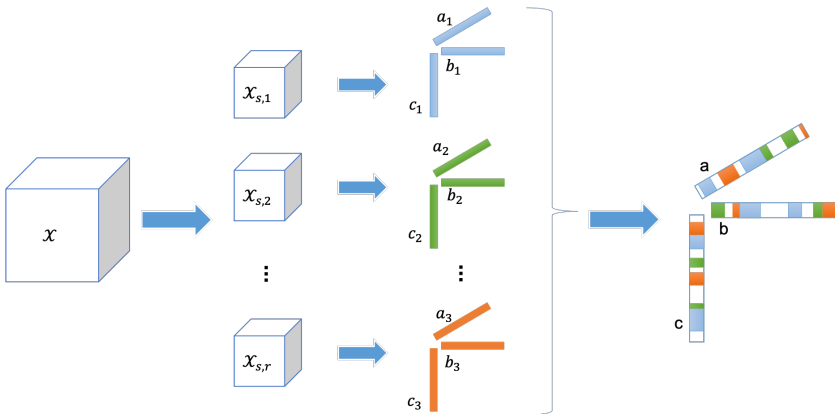

Figure 2.5: Illustration of ParCube

Suppose a tensor **X** is of size $I \times J \times K$ and we want a **R**-rank decomposition of the tensor.The algorithm is composed of three steps: sampling, decomposition of sub-blocks and merging. First, using mode densities as bias, randomly select a set of $100p\%$ ($\mathbf{p} \in [0,1]$) indices I_p, J_p, K_p to be common across all repetitions. With a fixed set of indices, now sample with a sampling factor of **s** on the remaining indices, merge the newly sampled indices with the fixed set and perform a CP decomposition (CP_ALS or CP_NMU). Repeat this process for **r** repetitions. After r repetitions, merge the factors obtained from each individual decomposition. In order to determine the correct correspondence of columns between different factors, e.g. A_i, the algorithm calculates the inner product of the columns of A_1 and A_2, where an output of 1 indicates the common set of indices due to previous normalization on the common set. For details of the merging algorithm

or parCube in general, please refer to Appendix A or the original paper [13].

The key advantage of ParCube is the fact that on its first phase, it produces r independent tensors which are significantly smaller in size. Moreover, each sub-tensor can be consequently decomposed independently from the rest, and as a result, all r tensors can be decomposed in parallel (assuming that we have a machine with r cores). Apart from the benefit on computing speed, the algorithm tends to produce sparse outer-product approximations, which is a desirable property in many applications. For instance, ParCube produces over 90% sparser results than regular PARAFAC, while maintaining the same approximation error [13].

2.5 Smoothing

Smoothing attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal [15].

Tikhonov regularization is a popular smoothing technique used to regularize ill-posed problems [15] It solves ill-posed ordinary least squares by replacing the

original objective function $\|Ax-b\|^2$ with the following function,

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|^2+\lambda\|Lx\|^2$$

where the matrix L is referred to as the regularization operator and the scalar $\mu \geq 0$ as the regularization parameter [15]. Common choices of regularization operators for problems in one space-dimension are the identity matrix, as well as scaled finite difference approximations of a derivative, such as,

$$L_1 = \begin{bmatrix} -1 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix}, L_2 = \begin{bmatrix} 2 & -1 & 0 & \dots & \\ -1 & 2 & -1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -1 & 2 \end{bmatrix}$$

Tikhonov regularization is a generalization of ridge regression in machine learning, where the objective function of ridge regression is,

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|^2+\lambda\|x\|^2$$

where the matrix L in Tikhonov regularization is an identity matrix in ridge regression.

Chapter 3

Approach

The fact that the resulting factors of the CP model have abrupt spikes often hampers the accuracy of the model as well as the interpretability of the factors without advanced domain knowledge. In order to obtain more accurate and easily interpretable results, we would prefer the solution's components to be as smooth as possible. In this chapter, we propose modifications to known tensor factorization algorithms to improve smoothness of the decomposition factors. In section 1, we begin with smoothing regularization to CP decomposition. More specifically, we propose an extension of CP_OPT to solve this new problem formulation. Second, to scale to large datasets that may not fit into memory, we propose to replace the core decomposition of ParCube with CP_OPT_SMOOTH in section 2. In the last section of this chapter, in the case that the improvement of ParCube with CP_OPT_SMOOTH might be not significant, we also modify the sampling technique of ParCube with CP_ALS, so that we might obtain a smoother sample with more useful information.

3.1 CP_OPT_SMOOTH

In machine learning, ridge regression is used to regularize linear regression and prevent overfitting by punishing the loss function for high values of the coefficients, i.e. x. Ridge regression enforces the large coefficients to be lower, but it does not enforce the smaller ones to be zero. In other words, it will not get rid of irrelevant features but rather minimize their impact on the trained model. Similarly, for the purpose of our paper, in order to obtain smoother decomposition factors where the results are non-overlapping and distinct, we also want to penalize elements with large differences with its neighbors. To give more emphasis on the smoothness between each element and its neighbouring elements, we adopt Tikhonov regularization with a weighted matrix as the regularization matrix instead of the identity matrix in ridge regression.

For the ease of understanding the approach, we use L_1 from Chapter 2 as our L_i, such that it has -1 on the diagonals and 1 on the sub-diagonals. The product of $L_i\mathbf{A}^{(i)}$ gives the first-order row-wise difference for elements in each column. Therefore, the smoother each factor is, the smaller the magnitude of g is.

Adopting the Tikhonov regularization of ordinary least squares discussed in the previous chapter to tensor factorization yields the following new objective

function,

$$\min_{\hat{\mathscr{X}}} f = \underbrace{\frac{1}{2}||\mathscr{X} - \hat{\mathscr{X}}||_F^2}_{\text{original } f} + \underbrace{\frac{1}{2}\mu \cdot \sum_{i=1}^{N} \|L_i \mathbf{A}^{(i)}\|_F^2}_{g}$$
$$\text{s.t. } \hat{\mathscr{X}} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}; \ldots; \mathbf{A}^{(n)}]\!]$$

where $L_i \in \mathbb{R}^{n \times n}$, $\mathbf{A}^{(i)} \in \mathbb{R}^{n \times r}$, and $\mu \in \mathbb{R}^{+}$.

Choosing to solve our approach in the same manner as CP_OPT, a gradient-based algorithm where no close form solution is required, we take advantage of the flexibility of CP_OPT in its capability to incorporate many different versions of of regularization techniques, including the L_1 norm. Since the gradient of the original objective funtion is known, we only need to derive the gradient of g.

$$\frac{\partial g}{\partial \mathbf{a_r}^{(n)}} = L_r^T L_r \mathbf{A}^{(r)}$$

Now the gradient of the new objective function becomes

$$\frac{\partial f}{\partial \mathbf{a_r}^{(n)}} = -(\mathbf{Z} \bigtimes_{m=1, m \neq n}^{N} \mathbf{a_r}^{(m)}) + \sum_{l=1}^{R} \gamma_{\mathbf{rl}}{}^{(n)} \mathbf{a_l}^{(n)} + L_r^T L_r \mathbf{A}^{(r)}$$

and we could now pass the new function and its gradient to L-BFGS, to obtain the optimal objective value and the factors that would yield such optimum.

3.2 ParCube with CP_OPT_SMOOTH

The default algorithms used in ParCube are CP_ALS and CP_NMU. Although CP_OPT is more stable and more accurate than ALS or NMU, it is not adopted due to its slow speed which goes against the speedup purpose of ParCube. With the expectation that CP_OPT_SMOOTH will speed up CP_OPT, the goal is to adopt CP_OPT_SMOOTH in ParCube and explore how it works under the structure of ParCube.

3.3 ParCube_Neighbor

Since CP_OPT takes longer than CP_ALS or CP_NMU to converge, its disadvantage in its speed might grow further when CP_OPT is used in ParCube as the algorithm calls CP_OPT multiples times. The trade-off between the accuracy and stability over the running speed might no longer be worthwhile. Therefore, another option is to smooth the decomposition factors by smoothing the distribution of the sampled indices, rather than modifying the gradient.

In order to preserve the running speed, we propose to perform "neighbor padding" on both sides of the indices being sampled. For example, if the sampled indices are [3, 9], the list after padding with a bandwidth of 1 becomes [2, 3, 4, 8, 9, 10] and becomes [1, 2, 3, 4, 5, 7, 8, 9, 10, 11] with a bandwidth of 2.

However, if our goal is to sample n indices, the number of indices after padding is highly likely to exceed n and could be at many as $n \cdot (1+2 \cdot bandwidth)$ indices assuming no two indices are the same after padded. Therefore, in order to be in consistent with ParCube's original sampling initiative, we need to downsample from n to $\frac{n}{1+2 \cdot bandwidth}$ indices when we first sample. Notice that $\frac{size\_of\_total}{1+2 \cdot bandwidth} > 1$ is necessary, else all indices would be sampled. Yet, since it is likely that some indices already exist in the list while padding, the number of the indices after padding might not exceed the required number of n. In such case, simply randomly draw with mode density as bias from the remaining indices until the required number is reached as shown in Figure 3.1.

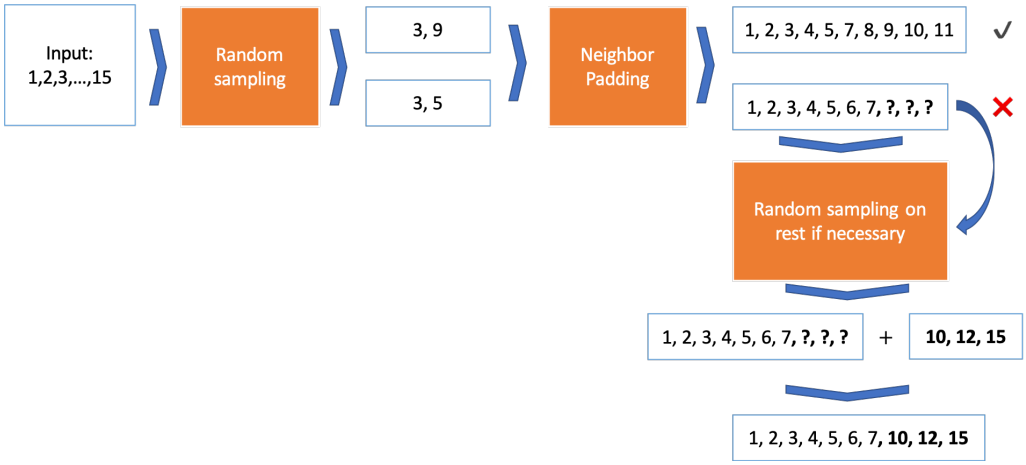

Figure 3.1: Illustration of neighbor padding

The neighbor padding process. With an input of 15 numbers, if we were to sample 10 numbers, the first random sampling samples 10/(2*2+1) = 2 numbers. Padding [3,9] does not involve overlapping numbers, but padding [3,5] does and it has less than 10 numbers after padding. The 3 missing numbers will be randomly sampled from the remaining numbers in the pool, i.e. [8, 9, 10, 11, 12, 13, 14, 15].

Since this approach only modifies the sampling technique, it will work with both default decomposition algorithms of ParCube, CP_ALS and CP_NMU as well as CP_OPT.

Chapter 4

Experiments

In this chapter, we evaluate our proposed methods on a real-life spatio-temporal dataset. We apply our approach to a real-life spatio-temporal dataset. Section 1 of this chapter will introduce the dataset and preprocessing needed for our experiment. Then, the following two sections will introduce the evaluation metrics and the baselines used for comparisons. Finally, section 5 presents the experimental results.

4.1 Data Description and Preprocessing

We evaluate our approaches on New York City Taxi Data provided by the NYC Taxi & Limousine Commission (TLC). The raw data contains over 4.5 million Uber pickups in New York City from April to September 2014. The Formidable Repository of Open Sparse Tensors and Tools (FROSTT) is a collection of publicly available sparse tensor datasets and tools [16]. FROSTT parses the raw data and

generates a publicly available sparse tensor dataset on NYC Uber pickups. The data is converted to a four-way tensor where each mode represents dates, hours, latitudes and longitudes and the tensor values are the pickup counts. Latitude and Longitude values are rounded to three decimal places (i.e., 110 meters of resolution). In other words, the area is divided into grids of 110 meter × 110 meter.

For the purpose of demonstration, we drop the date information and use a three-way tensor. For faster computation, we have generated a smaller tensor by dividing the spatial area into larger grids. Instead of having grids of 0.001 degree × 0.001 degree, we now use 0.0025 degree × 0.0025 degree, which is roughly 278 meter × 278 meter. The resulting tensor is of size 24 × 647 × 866.

4.2 Evaluation Metrics

The methods will be evaluated in terms of four different quantities: the fitness of the decomposition, the smoothness of elements in each factor, the sparsity of the factors and the computing time.

In order to evaluate how accurate the decomposition factors represent the original tensor, we measure the fitness F of a tensor factorization by calculating the norm difference between the approximated tensor $\hat{\mathscr{X}}$, reconstructed using

the decompostion factors $[\![\boldsymbol{\lambda};\mathbf{A}^{(1)};\ldots;\mathbf{A}^{(n)}]\!]$ and the original tensor $\mathscr{X}$,

$$\begin{aligned} F &= 1 - \frac{||\mathscr{X} - \hat{\mathscr{X}}||_F}{\|\mathscr{X}\|_F} \\ &= 1 - \frac{\sqrt{\|X\|^2 + \|\mathscr{X}\|_F^2 - 2\cdot < X \cdot \mathscr{X} >}}{\|\mathscr{X}\|_F} \end{aligned}$$

where $< \cdot >$ denotes the inner product of two tensors.

The smoothness G of the decomposition factors is computed using the concepts behind Tikhonov regularization function,

$$G = \sum_{i=1}^{N} \|L_i \mathbf{A}^{(i)}\|_F^2$$

As $L_i\mathbf{A}^{(i)}$ calculates the row-wise first order difference for each column, the smaller the norm is, the smoother the factors are. Notice that one of the goals of this paper is to explore smoothing techniques in factorization algorithms. As smoothing approximates a rough distribution of the data, we do anticipate a slight reduction in the fitness in exchange for a smaller smoothness.

Lastly, we also want to evaluate each method on its computing time. We anticipate improvement in time for both CP_OPT_SMOOTH and ParCube with CP_OPT_SMOOTH. With a smoothing regularization parameter μ, the objective

function f should converge faster, especially when μ is high. As for ParCube with CP_OPT_SMOOTH, the run time is expected to be fast as well due to the fact that the algorithm can be performed in parallel computing.

For each algorithm, we report the average fit, average smoothness, average sparsity and average run time. Each result is reported after 10 iterations of the algorithm. In order to avoid the underestimation brought by the instability of certain algorithms, each average result is calculated after dropping the lowest fit among the 10 iterations.

4.3 Results

4.3.1 Smoothing on CP_OPT

Baseline: CP_OPT

Since our first approach is to adapt smoothing to the stable and accurate gradient-based algorithm CP_OPT, we need to use results from CP_OPT factorization as our benchmark.

The only parameter involved in CP_OPT is the number of ranks r of the decomposition factors. In order to determine an r that will be used across the rest of the algorithms evaluated in this paper, we tune r for r in $[5, 10, 15, 20, 25, 30]$. Figure 4.1 shows the average fit F and average running time T for each r.

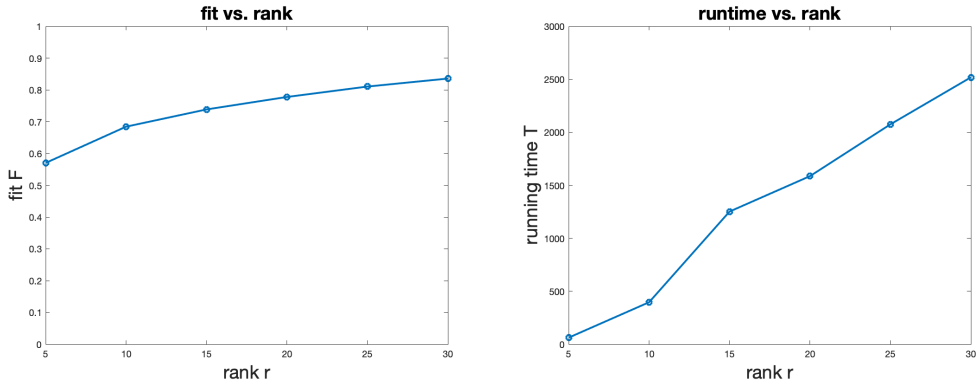

Figure 4.1: Plots of CP_OPT

The average fit F and average running time T of CP_OPT for r in $[5,10,15,20,25,30]$. The plot on the left shows F *vs.* r, the plot on the right shows T *vs.* r.

Notice that as r increases, the fit F improves because a higher-rank decomposition captures more underlying structures of the original tensor with more elements. However, the results in Figure 4.1 also shows that the improvement in F becomes less and less obvious as r increases. On the other hand, T also increases with r, but the amount of increase in T does not produce a significant improvement in F. Thus, compromising between the fit F and the computing time T, the results when $r = 15$ are chosen as the benchmark for later experiments. A more detailed results that include the smoothness G are shown in Table 4.1:

Rank r	Fit F	Smoothness G	Running Time T
5	0.5703	6.4949	62.6352
10	0.6842	15.1363	396.6122
15	**0.7385**	**23.4836**	**1251.9445**
20	0.7777	36.9896	1586.1656
25	0.8104	49.9276	2074.97938
30	0.8355	63.8163	2517.9802

Table 4.1: Detailed results of CP_OPT
Parameter tuning results of regular CP_OPT. The parameter and results in boldface will be used as the baseline for later comparisons.

CP_OPT_SMOOTH

As mentioned in section 4.3.1, we compare our algorithm with the original CP_OPT approach at $r = 15$. The additional parameter in our modified approach is the regularization parameter μ. We tune our algorithm CP_OPT_SMOOTH for each μ in $\mu = [100, 10000, 25000, 50000, 75000, 100000]$. The plots in Figure 4.2 shows the average fit f and average smoothness G for different regularization parameters μ.

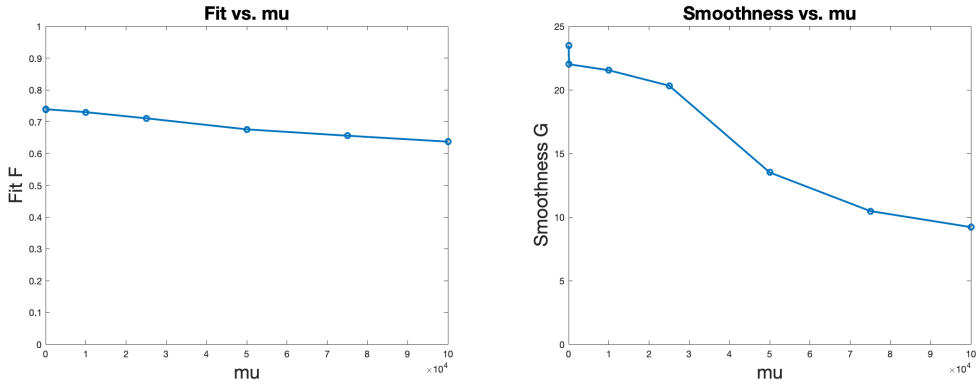

Figure 4.2: Plots of CP_OPT_SMOOTH
The average fit and average smoothness for different regularization parameters μ. Plot on the left: $F\ vs.\ \mu$; plot on the right:$G\ vs\ \mu$

From Figure 4.2, we could infer that there is barely any regularization power when $\mu < 20000$ as the fit F and the smoothness G have not deviated much from the baseline. Yet, there is a significant improvement in G when $\mu = 50000$ while the sacrifice in F is tolerable. When $\mu > 50000$, the changes in F and G are not as conspicuous as the changes we observe at $\mu = 50000$. Therefore, we conjecture that $\mu = 50000$ is the key turning point for smoothing when $r = 15$.

On the other hand, we observe a significant reduction in the running time T of CP_OPT_SMOOTH in Figure 4.3 as μ increases, which meets our assumption that smoothing would improve the computing time as the algorithm converges faster when the regularization parameter μ becomes larger.

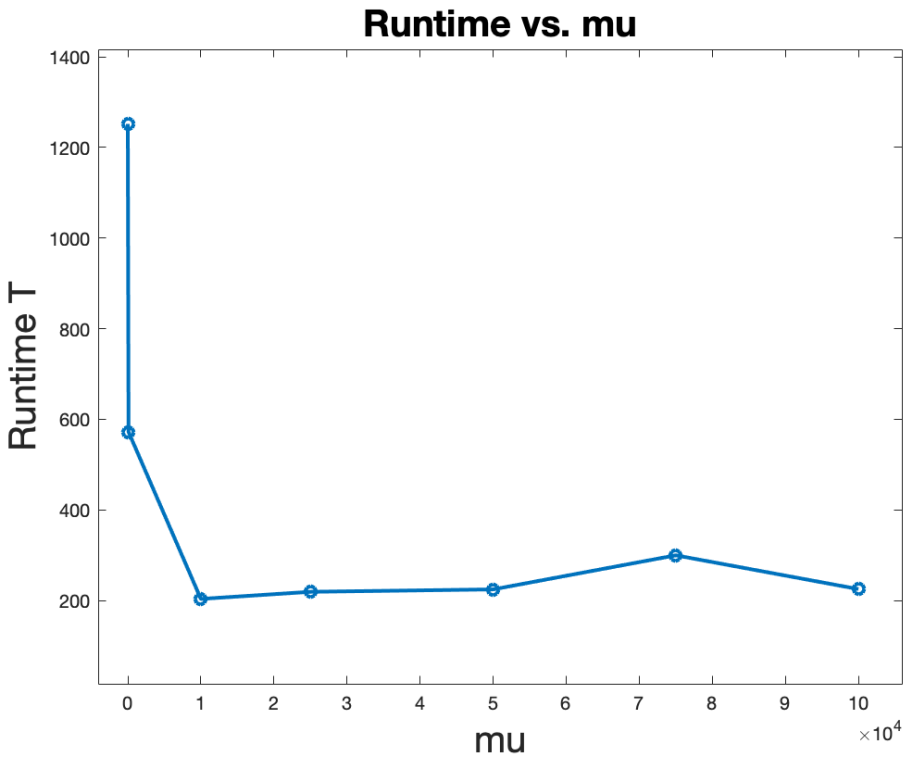

Figure 4.3: The average running time at different μ

Overall, at $r = 15$, we observe that CP_OPT_SMOOTH sacrifices some fitness in exchange of significant improvements on the smoothness of the factors and the running time of the algorithm. A more detailed results of CP_OPT_SMOOTH are shown in Table 4.2:

μ	Fit F	Smoothness G	Running Time T
0 (regular)	*0.7385*	*23.4836*	*1251.9445*
100	0.7391	22.0135	572.0885
10000	0.7299	21.5393	203.6456
25000	0.7106	20.3266	219.3096
50000	**0.6758**	**13.5053**	**224.6507**
75000	0.6558	10.4718	299.8607
100000	0.6371	9.2117	225.3254

Table 4.2: Detailed results of CP_OPT_SMOOTH
Parameter tuning results of CP_OPT_SMOOTH. The parameter and results in italic are the baseline results from regular CP_OPT. The parameter and results in boldface are the optimal trade-off results.

With Table 4.2, we conjecture that CP_OPT_SMOOTH is a substitute for CP_OPT when the scenario focuses more on smoothness and running time rather than simply the fitness. To verify this idea, we also experimented our approach on different ranks, when $r = [5, 10, 20, 25]$. Here, we only show visualization of the results on half of the previously used regularization parameters, $\mu = [100, 10000, 25000]$. For complete result table and plots, please refer to the Appendix B.

The plots in Figure 4.4 show the average fit and smoothness of CP_OPT_SMOOTH at different ranks with different magnitudes of the regularizing parameter.

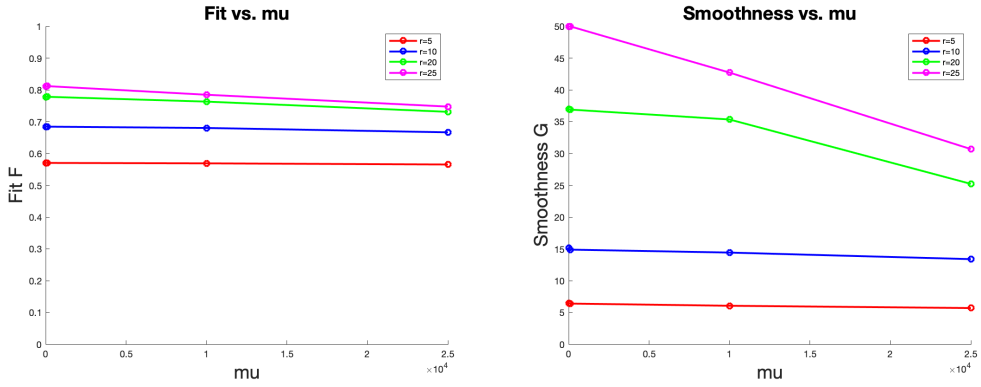

Figure 4.4: Plots of CP_OPT_SMOOTH at different r

The average fit and average smoothness for different regularization parameters μ and different ranks. The lines in red, blue, green and magenta are when r = 5, 10, 20, 25, respectively. Plot on the left: $F\ vs.\ \mu$; plot on the right:$G\ vs\ \mu$.

As shown in Figure 4.5, except for $r = 5$ where there is an increase in the running time at first, we observe an immediate decline in the running time of CP_OPT_SMOOTH for all other ranks starting from when $\mu = 100$. We thus confirm the effectiveness in reducing computation time of CP_OPT.

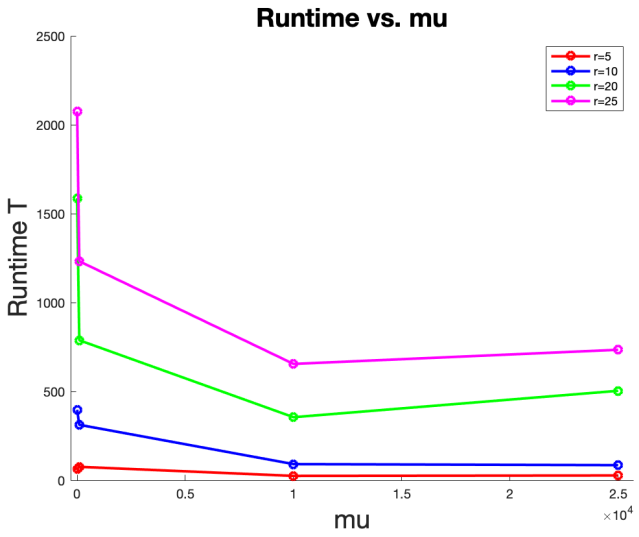

Figure 4.5: Runtime of CP_OPT_SMOOTH at different r
The average running time of rank r = [5, 10, 20, 25] at μ = [0, 100, 10000, 25000].

The complete results are shown in Table 4.3. Overall, although when both the rank r and the regularization parameter μ are small ($\mu < 10000$), the decrease in F as shown in Table 4.3 is minimal, we do indeed observe that as r increases, the impact of the same regularization parameter grows and the deviations in F and G become relatively more noticeable. In applications where smoothness is preferred, a trade-off between fit and smoothness is acceptable, particularly when there is a reduction in the running time of the algorithm.

r	μ	F	G	T
5	0	0.5703	6.4949	62.6352
5	100	0.5702	6.3975	75.7730
5	10000	0.5689	6.0561	24.8119
5	25000	0.5653	5.6967	27.3519
10	0	0.6842	15.1363	396.6122
10	100	0.6842	14.8822	312.1898
10	10000	0.6801	14.4166	91.1463
10	25000	0.6664	13.3855	85.2114
20	0	0.7777	36.9896	1586.1656
20	100	0.7781	36.8996	787.8983
20	10000	0.7630	35.3419	354.9436
20	25000	0.7309	25.2179	503.4650
25	0	0.8104	49.9276	2074.97938
25	100	0.8117	49.9798	1232.2461
25	10000	0.7846	42.7149	654.7238
25	25000	0.7473	30.6858	734.2529

Table 4.3: Experimental results of CP_OPT_SMOOTH at different ranks r = [5, 10, 20, 25].

4.3.2 ParCube with CP_OPT_SMOOTH

Baseline: ParCube with CP_OPT

The result in the previous section demonstrates the ability of CP_OPT_SMOOTH to significantly speed up CP_OPT and smooth the factors at the cost of some loss in accuracy. The improvement on the calculating speed immediately becomes

convenient for the second approach of our study, which is to adopt CP_OPT or its modified version to ParCube. As introduced previously, the default CP methods in ParCube are CP_ALS and CP_NMU since CP_OPT is slow, but now we might be able to apply CP_OPT to ParCube with CP_OPT_SMOOTH. In order to evaluate the performance of ParCube with CP_OPT_SMOOTH, we will compare its performance with CP_OPT, CP_OPT_SMOOTH, and ParCube with CP_OPT.

The parameters involved in ParCube include the fixed ratio p that determines the percentage of the indices to be shared across all sub_tensors, the sample factor s that indicates each sub_tensor is of the size $\frac{I}{s} \times \frac{J}{s} \times \frac{K}{s}$, and $times$ that implies the number of repetitions of factorizing on sub_tensors. As suggested in the original paper, a reasonable value of p is about 10%-20%[13], but in practice, $p = 55\%$ is commonly used across all ParCube models. We tune the parameters of the baseline model on $[s, times]$ using the cartesian product of $s = [1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]$ and $times = [5, 10, 15]$. An example of the results when the sample factor $s = 2$ is shown in Table 4.4:

r	F	G	T
5	-1.1034	6.4189 e8	87.1683
10	-1.2684	4.8778 e8	521.2419
15	-1.8014	1.0773 e10	1608.2130

Table 4.4: Results of regular ParCube with CP_OPT when s = 2

Although we did expect that the running time would be significantly slower than any other algorithms discussed previously, we did not expect the negative fits as shown in the table above. Furthermore, we observe that the smoothness G of ParCube has a significantly larger scale than the ones shown previously, but this makes sense as the nature of ParCube with random sub-sampling promotes sparsity.

ParCube with CP_OPT_SMOOTH

The additional parameter that ParCube with CP_OPT_SMOOTH has is the regularization parameter μ. Here, we arbitrarily choose $\mu = 10000$ based on observations from the complete table of results in Appendix B. Since we observe that there typically is a significant improvement in smoothness when $\mu = 50000$ and ParCube's scheme samples only a fraction of the original data, we therefore arbitrarily shrink the optimal $\mu = 50000$ to 10000 so that the regularization strength will not completely over-shadow the original fitting objective.

The result of ParCube with CP_OPT_SMOOTH is shown in Table 4.5:

r	F	G	T
5	0.4753	1.4018 e8	32.4673
10	0.4844	1.8111 e8	135.2103
15	0.4610	1.6706 e8	412.4211

Table 4.5: Results of regular ParCube with CP_OPT_SMOOTH when s = 2

With CP_OPT_SMOOTH, we manage to obtain positive and reasonable fits rather than the negative fits obtained through ParCube with CP_OPT. The smoothness appears to be much smaller but it might not be reasonable to compare these results with the ones shown in Table 4.4 where all the corresponding fits were negative. Furthermore, since our focus of ParCube with CP_OPT_SMOOTH is its performance in terms of accuracy and runtime, we compare these values against those of CP_OPT and CP_OPT_SMOOTH.

As shown in Table 4.6, although we have managed to improve the result of ParCube with CP_OPT using CP_OPT_SMOOTH, compared to simple, non-parallel methods, such as CP_OPT or CP_OPT_SMOOTH, we observe obvious decrease in fit as shown in Table 4.6. Furthermore, Table 4.5 shows that we observe obvious decrease in speed when compared to CP_OPT, but the running time did not outperform that of CP_OPT_SMOOTH. However, our results are computed in parallel on a machine with 4 cores. As the algorithm is designed to run in parallel, the running time would outperform CP_OPT_SMOOTH when the ParCube approach is ran on a machine with more computing power.

Overall, with the speedup benefit from CP_OPT_SMOOTH, we were able to implement a smoother CP_OPT in ParCube. The new approach loses a portion of accuracy for a faster runtime than CP_OPT, and a faster runtime than

CP_OPT_SMOOTH if ran on a machine with high computing power.

r	CP_OPT	CP_OPT_SMOOTH	ParCube with CP_OPT_SMOOTH
5	0.5703	0.5594	0.4753
10	0.6842	0.6505	0.4844
15	0.7384	0.6758	0.4610

Table 4.6: Results of fit F for CP_OPT, CP_OPT_SMOOTH and ParCube with CP_OPT_SMOOTH

r	CP_OPT	CP_OPT_SMOOTH	ParCube with CP_OPT_SMOOTH
5	62.6352	28.9384	32.4673
10	396.6122	85.2512	135.2103
15	1251.9445	224.6507	412.4211

Table 4.7: Results of runtime T for CP_OPT, CP_OPT_SMOOTH and ParCube with CP_OPT_SMOOTH

4.3.3 ParCube_Neighbor

Different from the approach above, ParCube_Neighbor tries to obtain smoother result factors in ParCube through smoothing its sampling method. Since our dataset represents the number of Uber pickups in an area, it makes more sense to explore our approach using ParCube with CP_NMU than with CP_ALS. Therefore, we use the performance of regular ParCube with CP_NMU as the baseline, and compare the performance of our approach with it.

Similar to our experiments in the previous approach, We tune the param-

eters of the baseline model on $[s, times]$ using the cartesian product of $s = [1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]$ and $times = [2, 4, 6, 8, 10]$. An example of the results when the sample factor $s = 2$ and $s = 2.5$ is shown in Figure 4.6 and Figure 4.7:

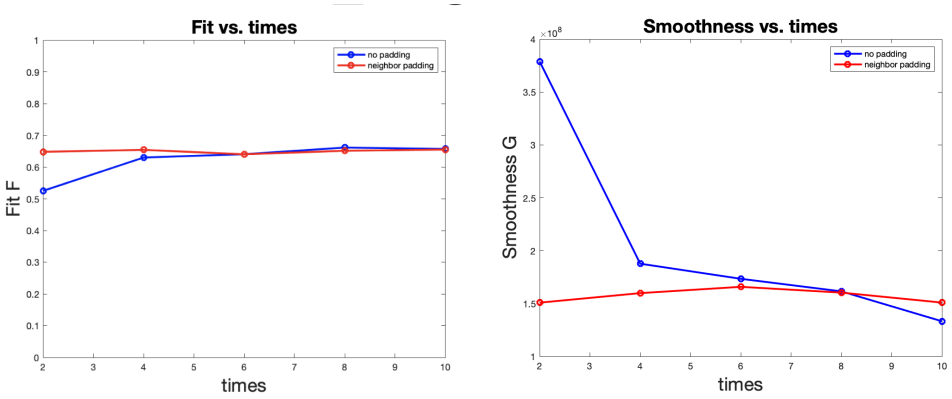

Figure 4.6: Padding vs. Non-padding at s = 2

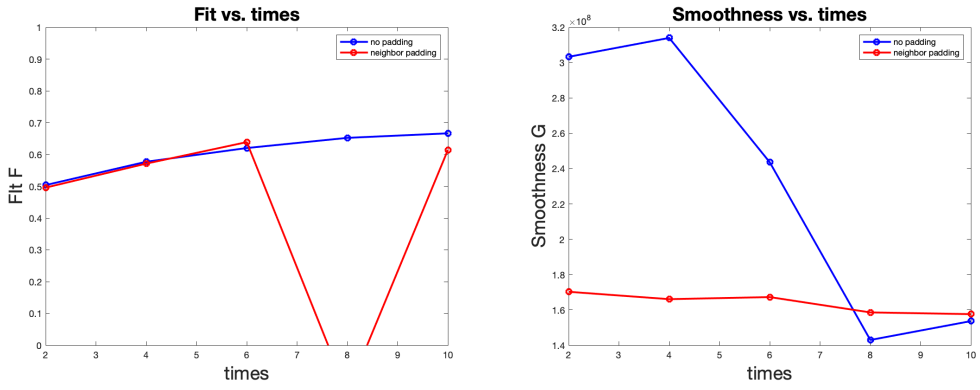

Figure 4.7: Padding vs. Non-padding at s = 2.5

With the red line denoting the performance of the padded ParCube and the blue the original approach, we easily observe that at $s = 2$, ParCube_Neighbor

has an improvement in fit when the number of repetions $times$ performed in ParCube is low and maintains the same accuracy when $times$ is high. We could almost reach the same conclusion from Figure 4.7 where we observe that ParCube with padding mostly maintains the same fit level, except for the big drop of negative at $times = 8$. This might have happened since CP_NMU, as an extension of CP_ALS, is also not stable and do not guarantee to converge to a global minimum or a stationary point. Such unstability might have caused a bad approximation of the original tensor, which further results in a negative fit.

In terms of smoothness, which is our priority focus of the paper, we observe that there are significant improvements in smoothness until $times = 6$. This might have happened because at $times = 6$, we have sampled the majority of the indices that contain important information of the tensor, padding or not would not make an obvious difference in the decomposition. Overall, ParCube_Neighbor significantly improves smoothness while maintaining the same level of accuracy.

Chapter 5

Conclusion

With the convenience of tensor representation of spatio-temporal data, tensor factorization has become the popular approach people take to analyze spatio-temporal data. Yet, as many tensor factorization methods produce results that are highly non-smooth, advanced domain knowledge is still often required to further interpret each factor. In order to obtain more accurate and easily interpretable results, we would prefer the solution's components to be as smooth as possible. Whereas earlier works have been focusing on promoting non-overlapping results for a specific, fixed tensor mode [18], which would still require prior domain knowledge, we propose three approaches in promoting multi-modal smooth decomposition

5.1 Current Work

The first approach CP_OPT_SMOOTH achieves smoother factors by imposing Tikhonov regularization in the objective function of CP. Although we lose

some accuracy as the regularization strength increases, we not only observe improvement in smoothness but also a significant drop in the running time of the algorithm. Especially when the regularization parameter is small, we observe that the algorithm maintains roughly the same fit and smoothness at a much faster speed.

The benefit of the first approach allows us to move on to our second approach, which is to adopt CP_OPT, an algorithm more stable and accurate than CP_NMU or CP_ALS but much slower, to ParCube, a parallelizable factorization method designed to achieve high accuracy at low complexity. As we have managed to incorporate CP_OPT_SMOOTH as a substitute for CP_OPT in ParCube, this algorithm sacrifices quite some accuracy when compared to simple CP_OPT. Furthermore, this algorithm runs faster than CP_OPT but slower than CP_OPT_SMOOTH when ran on a 4-core machine. Yet, as the algorithm is designed to perform in parallel, we do expect a significant reduction in computing time when using a higher computing power machine.

Lastly, our approach to smooth the sampled indices in ParCube was successful when experimenting with CP_NMU. We observe that the algorithm significantly improves smoothness of factors while achieving the same accuracy without smoothing applied.

5.2 Future Directions

The successful result we observe from the third approach ParCube_Neighbor experimented with CP_NMU brings out the question of whether this method can improve the smoothness of any type of ParCube. Ideally, since our approach only modifies the sampling method in ParCube, it should be applicable to ParCube with CP_ALS as well. More experimental results need to be done to confirm or reject this idea.

Second, as CP_OPT is more stable and accurate than the other two algorithms, it is more ideal to incorporate ParCube with CP_OPT. As we have managed to make it possible by reducing its runtime with CP_OPT_SMOOTH, combing ParCube_Neighbor with CP_OPT_SMOOTH could potentially produce a more stable and much smoother outcome.

Appendix A

ParCube Algorithms

Algorithm 1 BiasedSample

1: **Input:** Original tensor $\mathcal{X}$ of size $I \times J \times K$, sampling factor s.

2: **Output:** Sampled tensor $\hat{\mathcal{X}}$, index sets $\mathcal{I}, \mathcal{J}, \mathcal{K}$.

3: Compute

$$x_a(i) = \sum_{j=1}^{J} \sum_{k=1}^{K} \mathcal{X}_{\rangle,|,\|}, x_b(i) = \sum_{i=1}^{I} \sum_{k=1}^{K} \mathcal{X}_{\rangle,|,\|}, x_c(i) = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathcal{X}_{\rangle,|,\|}.$$

4: Compute set of indices $\mathcal{I}$ as random sample without replacement of $\{1, ..., I\}$of size I/s with probability $p_{\mathcal{I}}/(\sum_{i=1}^{I} x_a(i)$. Likewise for $\mathcal{J}, \mathcal{K}$.

5: Return $\hat{\mathcal{X}} = \mathcal{X}_{\mathcal{I},\mathcal{J},\mathcal{K}}$

Algorithm 2 BasicParCube

1: **Input:** Tensor $\mathcal{X}$ of size $I \times J \times K$, rank r, sampling factor s.

2: **Output:** Factor matrices A, B, C of size $I \times F, J \times F, K \times F$, respectively.

3: Run Biased Sample($\mathcal{X}$, s) (Algorithm 1)

4: Run CP_ALS and obtain $\hat{\mathcal{X}}_f$ and $\mathcal{I}, \mathcal{J}, \mathcal{K}$.

5: $A(\mathcal{I},:) = A_s, B(\mathcal{J},:) = B_s, C(\mathcal{K},:) = C_k$.

Algorithm 3 ParCube

1: **Input:** Tensor $\mathscr{X}$ of size $I \times J \times K$, rank r, sampling factor s, number of factors r.

2: **Output:** PARAFAC factor matrices A, B, C of size $I \times F, J \times F, K \times F$, respectively, and vector λ of size $r \times 1$ which contains the scale of each component.

3: Initialize A, B, C to all-zeros.

4: Randomly, using mode densities as bias, select a set of 100p% ($p \in [0,1]$) indices I_p, J_p, K_p to be common across all repetitions.

5: **for** i = $1, \ldots, R$ **do**

6: Run Algorithm 2 with sampling factor s, using I_p, J_p, K_p as a common reference among all r different samples and obtain A_i, B_i, C_i. The sampling is made on the set difference of the set of all indices and the set of common indices.

7: Calculate the l2 norm of the columns of the common part: $n_a(r) = \|A_i(I_p, r)\|_2$, $n_b(r) = \|B_i(J_p, r)\|_2$, $n_c(r) = \|C_i(K_p, r)\|_2$ for $r = 1 \ldots R$. Normalize columns of A_i, B_i, C_i using n_a, n_b, n_c and set $\lambda_i(r) = n_a(r) n_b(r) n_c(r)$. Note that the common part will now be normalized to unit norm.

8: **end for**

9: A =FactorMerge (A_i), B =FactorMerge (B_i),C =FactorMerge (C_i)

10: λ = average of λ_i.

Algorithm 4 FactorMerge

Input: Factor matrices A_i of size $I \times F$ each, where $i = 1 \dots r$, and r is the number of repetitions, I_p: set of common indices.
Output: Factor matrix A of size $I \times F$.
Set $A = A_1$
for i = 2, ..., r **do**
  for $r_1 = 1, \dots, R$ **do**
    for $r_2 = 1, \dots, R$ **do**
      Compute similarity $v(r_2) = (A(I_p, r_2))^T (A_i(I_p, r_1)))$
    end for
    $c = \arg\max_{c'} v(c')$
    Update only the zero entries of $A(:, c)$ using vector $A_i(:, r1)$.
  end for
end for

Appendix B

Complete Result of CP_OPT_SMOOTH

r	μ	F	G	T
5	0	0.5703	6.4949	62.6352
5	100	0.5702	6.3975	75.7731
5	10000	0.5689	6.0562	24.8120
5	25000	0.5653	5.6968	27.3519
5	50000	0.5594	5.2572	28.9384
5	75000	0.5551	4.9018	30.4896
5	100000	0.4915	3.8446	26.0330
10	0	0.6842	15.1363	396.6122
10	100	0.6843	14.8823	312.1898
10	10000	0.6801	14.4167	91.1464
10	25000	0.6665	13.3856	85.2115
10	50000	0.6505	11.7261	85.2512
10	75000	0.6349	10.4533	94.6779
10	100000	0.6178	8.5544	106.1122
20	0	0.7777	36.9896	1586.1656
20	100	0.7781	36.8996	787.8983
20	10000	0.7630	35.3419	354.9436
20	25000	0.7309	25.2180	503.4651
20	50000	0.6914	15.8321	522.4559
20	75000	0.6695	11.7956	472.7661
20	100000	0.6477	9.5556	412.0116
25	0	0.8104	49.9276	2074.97938
25	100	0.8118	49.9798	1232.2461
25	10000	0.7846	42.7150	654.7238
25	25000	0.7474	30.6859	734.2529
25	50000	0.7027	17.8443	1053.5411
25	75000	0.6706	12.0774	762.5556
25	100000	0.6537	9.8782	716.4697
30	0	0.8355	63.8163	2517.9802
30	100	0.8365	63.0155	1863.5534
30	10000	0.8022	48.5213	1469.4938
30	25000	0.7576	34.9737	1507.8197
30	50000	0.7121	20.8218	1849.7939
30	75000	0.6781	12.7963	1299.6351
30	100000	0.6540	10.0304	945.0380

Bibliography

[1] Ardavan Afshar, Joyce C. Ho, Bistra Dilkina, Ioakeim Perros, Elias B. Khalil, Li Xiong, and Vaidy Sunderam. Cp-ortho: An orthogonal tensor factorization framework for spatio-temporal data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, pages 67:1–67:4, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5490-5. doi: 10.1145/3139958.3140047. URL http://doi.acm.org/10.1145/3139958.3140047.

[2] J. Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, Sep 1970. ISSN 1860-0980. doi: 10.1007/BF02310791. URL https://doi.org/10.1007/BF02310791.

[3] Henry Crosby, Paul Davis, Theo Damoulas, and Stephen A. Jarvis. A spatio-temporal, gaussian process regression, real-estate price

predictor. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPACIAL '16, pages 68:1–68:4, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4589-7. doi: 10.1145/2996913.2996960. URL http://doi.acm.org/10.1145/2996913.2996960.

[4] Richard A. Harshman, Peter Ladefoged, H. Graf von Reichenbach, Robert I. Jennrich, Dale Terbeek, Lee Cooper, Andrew L. Comrey, Peter M. Bentler, Jeanne Yamane, and Diane Vaughan. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. 1970.

[5] T. Henretty, M. Baskaran, J. Ezick, D. Bruns-Smith, and T. A. Simon. A quantitative and qualitative analysis of tensor decompositions on spatiotemporal data. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sep. 2017. doi: 10.1109/HPEC.2017.8091028.

[6] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009. doi: 10.1137/07070111X.

[7] Joseph B. Kruskal. Three-way arrays: rank and uniqueness of trilinear

decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2):95 – 138, 1977. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(77)90069-6. URL http://www.sciencedirect.com/science/article/pii/0024379577900696.

[8] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. ISSN 1476-4687. doi: 10.1038/44565. URL https://doi.org/10.1038/44565.

[9] Haiping Lu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, 44(7):1540 – 1551, 2011. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2011.01.004. URL http://www.sciencedirect.com/science/article/pii/S0031320311000136.

[10] Daniel M. Dunlavy, Tamara Gibson Kolda, and Evrim Acar. Cpopt : optimization for fitting candecomp/parafac models. 01 2008.

[11] M. Mørup. Applications of tensor (multiway array) factorizations and decompositions in data mining, 2011.

[12] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data val-

ues. *Environmetrics*, 5(2):111–126, 1994. doi: 10.1002/env.3170050203. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/env.3170050203.

[13] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 521–536, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[14] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. Technol.*, 8(2):16:1–16:44, October 2016. ISSN 2157-6904. doi: 10.1145/2915921. URL http://doi.acm.org/10.1145/2915921.

[15] Lothar Reichel and Qiang Ye. Simple square smoothing regularization operators. *Electronic Transactions on Numerical Analysis. Volume*, 33:63–83, 01 2009.

[16] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. FROSTT: The formidable repository of open sparse tensors and tools, 2017. URL http://frostt.io/.

[17] Donghui Wang and Shu Kong. Feature selection from high-order tensorial data via sparse decomposition. *Pattern Recognition Letters*, 33(13):1695 – 1702, 2012. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2012.06.010. URL http://www.sciencedirect.com/science/article/pii/S0167865512001985.

[18] Yichen Wang, Robert Chen, Joydeep Ghosh, Joshua C. Denny, Abel N Kho, You Chen, Bradley A. Malin, and Jimeng Sun. Rubik: Knowledge guided tensor factorization and completion for health data analytics. In *KDD 2015 - Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 2015-August, pages 1265–1274. Association for Computing Machinery, 8 2015. doi: 10.1145/2783258.2783395.

[19] J. Xu, D. Deng, U. Demiryurek, C. Shahabi, and M. v. d. Schaar. Mining the situation: Spatiotemporal traffic prediction with big data. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):702–715, June 2015. ISSN 1932-4553. doi: 10.1109/JSTSP.2015.2389196.